2005

Neural Network Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle

Frederick G. Harmon
Cedarville University, fharmon@cedarville.edu

Recommended Citation

Harmon, Frederick G., "Neural Network Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle" (2005). *Engineering and Computer Science Faculty Publications*. 191.
http://digitalcommons.cedarville.edu/engineering_and_computer_science_publications/191

Neural Network Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle

by

FREDERICK G. HARMON

B.S. Electrical Engineering (Embry-Riddle Aeronautical University) 1992
M.S. Electrical Engineering (Air Force Institute of Technology) 1996

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Mechanical and Aeronautical Engineering

in the
OFFICE OF GRADUATE STUDIES
of the
UNIVERSITY OF CALIFORNIA
DAVIS

Approved: ______________________

____

____

Committee in Charge

2005

Disclaimer

MAY 19 2005

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 12.May.05	3. REPORT TYPE AND DATES COVERED DISSERTATION

4. TITLE AND SUBTITLE NEURAL NETWORK CONTROL OF A PARALLEL HYBRID-ELECTRIC PROPULSION SYSTEM FOR A SMALL UNMANNED AERIAL VEHICLE	5. FUNDING NUMBERS
6. AUTHOR(S) MAJ HARMON FREDERICK G	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) UNIVERSITY OF CALIFORNIA AT DAVIS	8. PERFORMING ORGANIZATION REPORT NUMBER CI04-1076
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) THE DEPARTMENT OF THE AIR FORCE AFIT/CIA, BLDG 125 2950 P STREET WPAFB OH 45433	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT Unlimited distribution In Accordance With AFI 35-205/AFIT Sup 1 DISTRIBUTION STATEMENT A Approved for Public Release Distribution Unlimited	12b. DISTRIBUTION CODE
13. ABSTRACT *(Maximum 200 words)*	

14. SUBJECT TERMS			15. NUMBER OF PAGES 280
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Frederick G. Harmon
June 2005
Mechanical and Aeronautical Engineering

Neural Network Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle

Abstract

Parallel hybrid-electric propulsion systems would be beneficial for small unmanned aerial vehicles (UAVs) used for military, homeland security, and disaster-monitoring missions. The benefits, due to the hybrid and electric-only modes, include increased time-on-station and greater range as compared to electric-powered UAVs and stealth modes not available with gasoline-powered UAVs. This dissertation contributes to the research fields of small unmanned aerial vehicles, hybrid-electric propulsion system control, and intelligent control. A conceptual design of a small UAV with a parallel hybrid-electric propulsion system is provided. The UAV is intended for intelligence, surveillance, and reconnaissance (ISR) missions. A conceptual design reveals the trade-offs that must be considered to take advantage of the hybrid-electric propulsion system. The resulting hybrid-electric propulsion system is a two-point design that includes an engine primarily sized for cruise speed and an electric motor and battery pack that are primarily sized for a slower endurance speed. The electric motor provides additional power for take-off, climbing, and acceleration and also serves as a generator during charge-sustaining operation or regeneration. The intelligent control of the hybrid-electric propulsion system is based on an instantaneous optimization algorithm that generates a hyper-plane from the nonlinear efficiency maps for the internal combustion engine, electric motor, and lithium-ion battery pack. The hyper-plane incorporates charge-depletion and charge-sustaining strategies. The optimization algorithm is flexible and allows the operator/user to assign relative importance between the use of gasoline, electricity, and recharging depending on the intended mission. A MATLAB/Simulink model was developed to test the control algorithms. The Cerebellar Model Arithmetic Computer (CMAC) associative memory neural network is applied to the control of the UAV's parallel hybrid-electric propulsion system. The CMAC neural network approximates the hyper-plane generated from the instantaneous optimization algorithm and produces torque commands for the internal combustion engine and electric motor. The CMAC neural network controller saves on the required memory as compared to a large look-up table by two orders of magnitude. The CMAC controller also prevents the need to compute a hyper-plane or complex logic every time step.

Frederick G. Harmon
June 2005
Mechanical and Aeronautical Engineering

Neural Network Control of a Parallel Hybrid-Electric Propulsion System for a Small Unmanned Aerial Vehicle

Abstract

Parallel hybrid-electric propulsion systems would be beneficial for small unmanned aerial vehicles (UAVs) used for military, homeland security, and disaster-monitoring missions. The benefits, due to the hybrid and electric-only modes, include increased time-on-station and greater range as compared to electric-powered UAVs and stealth modes not available with gasoline-powered UAVs. This dissertation contributes to the research fields of small unmanned aerial vehicles, hybrid-electric propulsion system control, and intelligent control. A conceptual design of a small UAV with a parallel hybrid-electric propulsion system is provided. The UAV is intended for intelligence, surveillance, and reconnaissance (ISR) missions. A conceptual design reveals the trade-offs that must be considered to take advantage of the hybrid-electric propulsion system. The resulting hybrid-electric propulsion system is a two-point design that includes an engine primarily sized for cruise speed and an electric motor and battery pack that are primarily sized for a slower endurance speed. The electric motor provides additional power for take-off, climbing, and acceleration and also serves as a generator during charge-sustaining operation or regeneration. The intelligent control of the hybrid-electric propulsion system is based on an instantaneous optimization algorithm that generates a hyper-plane from the nonlinear efficiency maps for the internal combustion engine, electric motor, and lithium-ion battery pack. The hyper-plane incorporates charge-depletion and charge-sustaining strategies. The optimization algorithm is flexible and allows the operator/user to assign relative importance between the use of gasoline, electricity, and recharging depending on the intended mission. A MATLAB/Simulink model was developed to test the control algorithms. The Cerebellar Model Arithmetic Computer (CMAC) associative memory neural network is applied to the control of the UAV's parallel hybrid-electric propulsion system. The CMAC neural network approximates the hyper-plane generated from the instantaneous optimization algorithm and produces torque commands for the internal combustion engine and electric motor. The CMAC neural network controller saves on the required memory as compared to a large look-up table by two orders of magnitude. The CMAC controller also prevents the need to compute a hyper-plane or complex logic every time step.

Acknowledgments

As I was preparing for the preliminary exams required for the PhD program, I read Psalm 102:25 in the Scriptures: "In the beginning you laid the foundations of the earth, and the heavens are the work of your hands." It reminded me that God miraculously created the universe. As I learn more about the world around us, the more I realize how little I know compared to our great and almighty God. He allows us to see glimpses of the order and design of the universe through science and engineering. I am thankful for the time that I have spent learning at UCD. I am most thankful for the love, faithfulness, and kindness of my Lord and Savior, Jesus Christ, who gave me the talents and abilities to earn the PhD degree.

My family has been very supportive and loving throughout my time here. [redacted] was patient with me when I needed to work during the day and late in the evenings. She managed the house well and kept up with the home schooling. The children were a much needed blessing. I am sure that [redacted] still has a draft copy of the dissertation somewhere in one of his treasure boxes. [redacted] drew and colored many pictures while sitting next to me at the computer desk. [redacted] put together many puzzles and played underneath my chair, and [redacted] liked to crawl, walk, and smile continuously. Our latest addition, [redacted], has also been a blessing in her first few months. Our family has enjoyed our time here, but we are looking forward to moving on to our next assignment.

A graduate program would be impossible to complete without the understanding, encouragement, and support of professors who have dedicated themselves to their work and the students. Professors Frank, Joshi, Chattot, Baughn, and Hess were instrumental in guiding me, giving me technical insight, and supporting me throughout my time at UCD. I appreciate their understanding to allow me to complete the PhD degree in the three years that I was allowed by the Air Force.

The fellow students in the Hybrid-Electric Vehicle (HEV) Center were very helpful during the dissertation work. Although I did not have time to work with the other students on the FutureTruck HEVs as I had hoped, their experience and help were invaluable. I wish the best for the other students such as Vern, Tom, Owen, Joey, and Chris. I know you all will do well wherever your careers take you.

Lastly, I am thankful for the career that God has given me in the Air Force. I never dreamed that I would make the military a career, but the opportunities have been tremendous. I am grateful that I was allowed to have three years to pursue a PhD here at UCD while still on active duty.

Table of Contents

List of Figures

List of Tables

Abstract

Parallel hybrid-electric propulsion systems would be beneficial for small unmanned aerial vehicles (UAVs) used for military, homeland security, and disaster-monitoring missions. The benefits, due to the hybrid and electric-only modes, include increased time-on-station and greater range as compared to electric-powered UAVs and stealth modes not available with gasoline-powered UAVs. This dissertation contributes to the research fields of small unmanned aerial vehicles, hybrid-electric propulsion system control, and intelligent control in the following areas:

1) **Conceptual Design of a Hybrid-Electric UAV:** A conceptual design of a small UAV with a parallel hybrid-electric propulsion system is provided. The UAV is intended for intelligence, surveillance, and reconnaissance (ISR) missions. The conceptual design reveals the trade-offs that must be considered to take advantage of the hybrid-electric propulsion system. The resulting hybrid-electric propulsion system is a two-point design that includes an engine primarily sized for cruise speed and an electric motor and battery pack that are primarily sized for endurance speed. The electric motor provides additional power for take-off, climbing, and acceleration and also serves as a generator during charge-sustaining operation or regeneration. Electric-only operation provides stealth operation not available with gasoline-powered UAVs by reducing the acoustic, smoke, and thermal signatures.

2) **Instantaneous Optimization of Energy Use:** The intelligent control of the hybrid-electric propulsion system is based on an instantaneous optimization algorithm (i.e. instantaneous rate of energy consumption) that generates a hyper-plane from the nonlinear efficiency maps for the internal combustion engine, electric motor, and lithium-ion battery pack. The hyper-plane incorporates charge-depletion and charge-sustaining strategies in addition to ideal operating band (IOB) and ideal operating line (IOL) concepts developed by previous researchers. The optimization algorithm is flexible and allows the user to assign relative importance between the use of gasoline, electricity, and recharging depending on the length and type of intended mission.

3) **Simulations using Intelligent Control Algorithm:** The Cerebellar Model Arithmetic Computer (CMAC) associative memory neural network is applied to the control of the UAV's hybrid-electric propulsion system. A MATLAB/Simulink model was developed to test the control algorithms. The CMAC neural network approximates the hyper-plane generated from the instantaneous optimization algorithm and produces torque commands for the internal combustion engine and electric motor. The CMAC neural network controller saves on the required memory as compared to a large look-up table by one to two orders of magnitude. The CMAC controller also prevents the need to compute a hyper-plane or complex logic every time step.

BLANK

Chapter 1: Overview

1.1 Introduction

Within the automotive industry, hybrid-electric vehicle (HEV) technology is leading to vehicles with increased fuel economy and reduced emissions. The same technology would have similar and even additional benefits if applied to unmanned aerial vehicles (UAVs) used for military, homeland security, and disaster-monitoring missions. A hybrid-electric vehicle is "a vehicle in which propulsion energy is available from two or more kinds or types of energy stores, sources, or converters, and at least one of them can deliver electrical energy" [1]. The task of controlling and optimizing the propulsion energy for a hybrid propulsion system is difficult due to the interaction of electrical, mechanical, thermodynamic, and electrochemical devices. Journal and conference papers describing control systems for hybrid-electric power trains and the optimization of the energy use began to appear in the 1980's and early 1990's. In the late 1990's and in this decade, papers appeared describing the application of adaptive control, optimal control, and fuzzy logic to the control of HEV power trains for automotive applications. A neural network controller is developed in this dissertation research to optimize the energy use of a parallel hybrid-electric propulsion system for a small UAV intended for military, homeland security, and disaster-monitoring missions.

HEV technology for automobiles can be applied to small or tactical UAVs to satisfy military missions. The potential benefits, due to the hybrid and electric-only modes, include increased endurance (time-on-station), longer range, stealth modes, and increased battery energy for the sensors. A hybrid-electric propulsion system for a UAV provides increased endurance time and longer range as compared to an electric-only powered UAV such as the current Dragon Eye or Desert Hawk [2]. The engine or fuel converter is down-sized for steady-state conditions and operated near a constant power output. The electric motor provides the additional power, if needed, for acceleration or climbing and serves as a generator at other times. Electric-only operation provides stealth operation not available with gasoline-powered UAVs and reduces the acoustic, smoke, and thermal signatures [3]. Also, electric-only operation eliminates exhaust emissions that interfere with chemical-detecting sensors. The already existing battery pack or generator that provides power for the video, electro-optic/infrared, or acoustic sensors now provides propulsion energy during certain phases of flight. In addition, the battery pack is available to power sensors while the UAV is setting in a surveillance location. Due to these advantages, a UAV with a hybrid-electric propulsion system enhances military-related missions.

The current military missions for a small or tactical UAV include force protection, surveillance, and reconnaissance. The Force Protection Aerial Surveillance System (FPASS) or Desert Hawk was designed for the Air Force security forces to conduct area surveillance, monitor runway approach and departure ends, and patrol base perimeters [2]. It was strongly needed to increase the security of overseas bases. Another small UAV, the electric-powered Dragon Eye, was built to conduct reconnaissance for the Marine Corps [2]. The Pioneer UAV, a tactical UAV, has been used by the Navy, Army, and Marine Corps for reconnaissance and surveillance. Wilson comments that "The Navy's Pioneer, a direct derivative of Israeli surveillance and reconnaissance UAVs, played a crucial role as a spotter for U.S. battleships. They were so effective that Iraqi troops began to associate the sound of the little aircraft's two-cycle engine with an imminent devastating bombardment" [4]. Additional military missions for the small or tactical UAV include intelligence, communications relay, chemical weapons detection, target acquisition, and battle-damage assessment.

A UAV with a hybrid-electric propulsion system could also be advantageous for homeland security missions such as pipeline inspection, seaport surveillance, and security for large facilities [4]. Electric-only operation (stealth mode) would prevent intruders from detecting the approaching UAV and would also prevent public noise disturbances while the UAV is flying over populated areas. The electric-only operation would also not cause any interference with highly-sensitive chemical or biological weapon sensors. The increased range and endurance time would also be beneficial for homeland security missions. Various government agencies have begun to realize these benefits and are considering hybrid-electric projects for UAV applications.

The Defense Advanced Research Projects Agency (DARPA), the NASA Glenn Research Center, and other government agencies are considering hybrid-electric propulsion systems for UAVs. DARPA's Micro Air Vehicle (MAV) project is designed to give the Army or Special Operations Forces a UAV with a reconnaissance and surveillance capability in a tactical environment [5]. The MAV is a vertical take-off and landing (VTOL) vehicle utilizing ducted fan technology. A series hybrid-electric propulsion system that includes a diesel engine, generator, electric motor, and batteries has been considered for the MAV. DARPA proposed that electric-only operation could provide a "perch and stare" capability. DARPA also has given a contract to Boeing to design a fuel cell-based hybrid-electric propulsion system for a UAV [6]. This Ultra Leap project was proposed to have military as well as civilian applications. An example of a civilian application is Helios, NASA's high-altitude, long-endurance UAV built by AeroVironment, designed for telecommunications and atmospheric monitoring [7]. The Helios currently uses solar power during the day to charge batteries giving it a limited night-time capability. A light weight fuel cell system is currently in the development stage that will provide an all-night capability. These projects in addition to several electric propulsion projects for general aviation aircraft illustrate that various organizations are evaluating hybrid-electric propulsion for aerospace applications [8, 9].

Another proposed application for a UAV with a hybrid-electric propulsion system is disaster monitoring such as the observation of forest fires. The California Space Institute is considering a hybrid-electric propulsion system for a small UAV intended for disaster monitoring. The hybrid-electric UAV would be useful for real-time monitoring of forest fires. Due to the cost of the sensors, the electric system would provide a back-up to the internal combustion engine while the UAV is operating in dense smoke conditions. The hybrid-electric system could potentially reduce the risk of losing expensive payloads and the UAV while it is operating in hazardous conditions.

Freeh of NASA recommends that for small or general aviation aircraft, "further investigation into various hybrid configurations may be beneficial based on both the technical and financial success of battery/IC engine automobile hybrids currently in the market" [9]. The researchers at UCD's HEV Center have been very successful in the design and development of parallel HEVs and have consistently placed in the top three in the FutureTruck competitions. This research applies the same technology to a small UAV to gain insight into the potential military uses of the technology.

1.2 Background

1.2.1 Objectives for the Hybrid-Electric UAV Propulsion System Controller

The three objectives of the advanced control system for the hybrid-electric unmanned aerial vehicle (HEUAV) propulsion system are to:

- Increase the range (corresponds to fuel economy)
- Provide adequate time for the UAV to operate in stealth (electric-only) mode
- Provide adequate battery power for the UAV's sensors (in the air or on the ground)

These overall objectives are directly related to several operational metrics listed in the Office of the Secretary of Defense's (OSD) UAV roadmap. The hybrid-electric propulsion system, depending on the mission, could potentially meet the capability metrics of a "30% increase in time-on-station requirement with the same fuel load" and "a UAV inaudible from a 500-1000 ft slant range" [2]. The FutureTruck vehicles designed by the HEV Center with a parallel hybrid-electric configuration have increased fuel economy and the same technology applied to a UAV results in increased range and endurance (time-on-station). The inaudible requirement, of course, is met during electric-only operation.

1.2.2 Hybrid-Electric Vehicle Configurations

The mechanical configuration of an HEV can be classified into two main categories: series and parallel as shown in Figure 1-1 [1, 10-13]. The internal combustion engine (ICE) in a series configuration acts as an auxiliary power unit (APU) to drive a generator that provides power to the energy storage system or the electric motor. Only the electric motor (EM) is connected to the mechanical drive train. The engine is not connected to the mechanical drive path which allows the engine to operate in an optimum torque and speed range independent of the driving conditions. However, large energy conversion losses exist between the mechanical and electrical system diminishing the overall system efficiency [14]. Also, the motor has to be sized for the maximum power required [10]. The series configuration is useful for low-speed, high-torque applications such as buses and aircraft tow tractors. In a parallel configuration, each energy source or converter can provide propulsion energy since the ICE and EM are both mechanically linked to the drive train. The torque of the electric motor can supplement the torque of the ICE, or if additional torque is available from the ICE, the ICE can operate the EM as a generator to recharge the battery pack. Because of the mechanical coupling, energy converters such as gas turbines with a relatively large turn on/off time cannot be used in the parallel configuration [12]. The speed of the drive train is not always the optimum speed for the engine, but the energy conversion losses are minimized. A continuously variable transmission (CVT), sometimes used in automotive applications, permits the engine to be operated near a constant speed and the electric motor permits the engine to operate close to a constant torque [14]. The engine and motor can be sized smaller than in a series configuration, and the motor is used as the generator so a separate generator is not required. The parallel configuration is used in most FutureTruck competition vehicles [15-17] and others such as the Honda Insight and Civic [18]. The parallel and series hybrid configurations are the traditional configurations, but others have been used such as the series-parallel configuration used in the Toyota Prius [1, 19] and the Nissan Tino [20]. This configuration has characteristics of both series and parallel configurations. The Toyota Prius uses a planetary gear with the generator attached to the sun gear, the engine connected to the planetary carrier, and the motor/axle to the ring gear [21]. This power split approach permits the engine to drive the generator and/or the motor/axle. The different configurations each have their advantages and disadvantages, and the application and the type of energy sources/converters often dictate the preferred configuration.

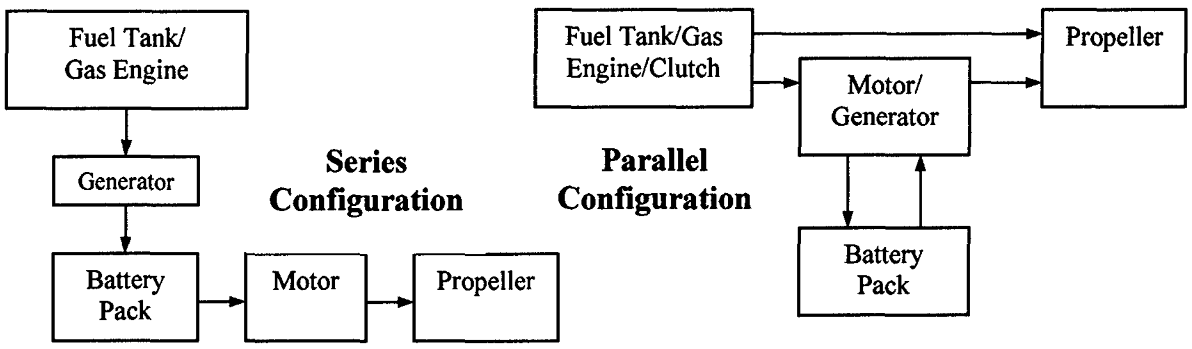


Figure 1-1: Series and Parallel Hybrid-Electric Configurations

An estimate was completed between a parallel and a series configuration for a small UAV (see Table 1-1). The estimate shows that the parallel configuration is lighter by approximately 2.5 lbs, or 8%, of the gross weight of 30 lbs. The extra weight for the series configuration is primarily due to the required generator and the larger electric motor. The series configuration and controller are mechanically and electronically simpler, but the disadvantages are the weight penalty and the energy conversion losses. Harmats also concluded that the parallel configuration was more effective than the series configuration for a hybrid-electric propulsion system (solar power/electric motor/engine) for a UAV [22]. The parallel configuration contains a more complicated controller and clutch/gearing mechanism, but weighs less which is a significant consideration for the UAV design. The parallel configuration also does not have the significant energy losses associated with the generator and battery charging/discharging. During flight, the parallel hybrid propulsion system allows the vehicle to be propelled directly with the ICE or the EM.

The parallel configuration is used for the HEUAV due to the advantages of a parallel configuration for analogous applications, the weight savings, and also because of the parallel HEV experience at UCD [16, 23, 24]. The parallel configuration permits a smaller engine and electric motor to be used as compared to the series configuration, and weight is saved since no APU/generator set is required. The advanced control system is designed to optimize the energy use of a parallel HEV configuration for the UAV application.

Table 1-1: Propulsion System Weight Comparison for Series and Parallel Hybrid-Electric Configurations for a 30 lb UAV

Component	**Parallel (Gasoline ICE+EM)**	**Series (Gasoline ICE +Generator+EM)**
Gasoline Engine[1]	2.5 (1.3 in^3)	3
48 oz. Fuel Tank	3	3
Generator[2]	N/A	1.5
Electric Motor[2]	0.5	1.5
Clutch/Gearing	0.5	N/A
Batteries	6	6
Total (lbs):	12.5	15

[1]The engine weight is estimated at 1.5 lbs/hp based on First Place Engine specifications [25]. A conventional 30 lb UAV would require a 35-45 cc gasoline engine weighing approximately 4.5 lbs.
[2]The electric motor and generator weights are estimated at 1 lb/hp based on Aveox electric motor specifications. A brushless DC motor has the highest power density [26].

1.2.3 Hybrid-Electric Vehicle Power Train Operating Strategies

In addition to the two primary HEV configurations, three overarching operating strategies are used for the energy management of an HEV: electric-only, charge-sustaining, and charge-depleting [27]. The electric-only strategy depends on the engine turn-on speed, the size of the battery pack, and the amount of low-speed operation. The electric-only strategy is available if the system is mechanically designed to permit it. The other two strategies are the hybrid approaches. The charge-sustaining hybrid strategy often uses a "thermostat" approach with an attempt to maintain the battery state-of-charge (SOC) at a certain level. This approach is often used for series hybrid-electric configurations [28] but is also used for parallel HEVs. The thermostat method allows the vehicle to be similar to conventional vehicles. If the engine is used to keep the battery at a specified SOC, the driver does not need to charge the battery pack from an external source and only needs to "fill up the gas tank." Another approach used for the charge-sustaining strategy is load-following where the output of the APU in the series hybrid is based on the rate of change of the battery SOC. In contrast to the charge-sustaining strategy, the charge-depleting strategy allows the battery SOC to decrease maximizing the energy use from off-board charging. This requires the driver to plug the vehicle into an external outlet to charge the batteries, hence the name "plug-in" HEV. The charge-depleting strategy can be used in series or parallel HEVs. The HEV Center's vehicles use a charge-depleting strategy until the battery pack drops to a low SOC and then a charge-sustaining strategy is used [16]. The charge-depleting strategy has been used effectively by the HEV Center in their plug-in vehicles, and the charge-sustaining strategy has been used successfully by Honda and Toyota in the marketplace.

The HEUAV uses a mix of the operating strategies depending on the mission and the intent of the operator. Due to the weight limitations, a relatively large battery pack and a purely charge-depleting strategy cannot be used. A purely charge-sustaining strategy will limit the endurance (time-on-station) and the stealth mode duration. Sufficient SOC is required for the enemy area or whenever the stealth (electric-only) mode is required. Because of this rationale, a combination of the charging strategies is used.

Two different modes are proposed for a relatively long surveillance mission: NORMAL or STEALTH. The NORMAL mode is the standard mode for take-off, climbing, cruising, and landing. The engine is used for take-off and climb with assistance from the motor, if needed, in a charge-depleting strategy. During cruise to the area of interest, a charge-sustaining strategy is used where the engine is primarily used for propulsion with margin for battery charging. During descent, the engine can be turned off and the motor used as a generator, similar to regenerative braking. The engine may also be idled back to avoid energy conversion losses associated with charging and discharging the batteries. During STEALTH operation, only the electric motor is used in an electric-only strategy. The two modes take advantage of the three operating strategies to maximize time-on-station in STEALTH mode and to increase the range.

1.2.4 Hybrid-Electric Vehicle Power Train Control Approaches

The operating strategies and modes are overarching approaches. For each of the strategies and modes, control algorithms are used to optimize the energy or power use of the propulsion system. In addition to rule-based or logic-based strategies, several advanced control approaches have been reported in the literature for the control of HEV power trains in automotive applications:

- Optimal Control
- Fuzzy Logic
- Neural Networks
- Adaptive Control

- Nonlinear Control
- Genetic Algorithms

These approaches have primarily been studied or applied in an academic environment or on research vehicles. A brief overview of each and their application to the control of HEV power trains are given in Chapter 2.

The goal of an advanced control system is to use a minimal amount of energy by finding the best combination of motor torque and engine torque as a function of power train speed, battery SOC, torque demand, or other parameters. Of the hybrid-electric power train advanced control schemes appearing in the current literature, those based on artificial neural networks (ANN) or fuzzy logic appear to be the most promising due to the relatively low computational resources needed and because an accurate power train propulsion model is not required (an accurate model is required for simulations). These approaches are also useful for nonlinear and multi-variable systems, can learn, and generalize. While results from many of the other control methods such as optimal control are very good, the computational requirements are too excessive for the embedded microcontrollers and the majority of the theory is for linear models.

The use of ANNs in HEV applications has been limited. Due to the potential benefits, this particular control method was selected for the HEUAV application. It is well known that ANNs can approximate nonlinear functions. An ANN has tremendous potential if applied to the control of the nonlinear HEUAV propulsion system. A specific type of neural network, the Cerebellar Model Arithmetic Computer or Cerebellar Model Articulation Controller (CMAC), was chosen for this application due to its rapid training time, practical hardware implementation, and low computational cost [29]. The CMAC is an alternative to the more common back-propagation multi-layer network [30]. The CMAC ANN, described in detail in Chapter 3, has been successfully applied to industrial applications, vibration control, robotic control, and fuel-injection systems [31-42].

1.3 Dissertation Objectives and Approach

The objective of the dissertation research was to design a CMAC neural network to control a parallel hybrid-electric propulsion system for a small UAV that:

- provides good performance for a nonlinear, multi-input plant
- takes into consideration the energy use of all components and not just the internal combustion engine or electric motor
- requires minimal computational and memory requirements to allow implementation in an embedded microprocessor

Based on speed and altitude profiles for typical intelligence, surveillance, and reconnaissance (ISR) missions, a CMAC ANN control algorithm was designed that minimizes the energy use of a nonlinear hybrid-electric propulsion system for a UAV intended for military, homeland security, and disaster-monitoring missions. The controller takes into consideration the efficiency of the ICE, EM, and the batteries. The controller requires minimal computational power to permit implementation in an embedded microprocessor for real-time operation. A simple rule-based controller was used to validate the HEUAV model and to obtain baseline results. The CMAC ANN controller simulation results were compared to the rule-based controller results.

A CMAC neural network controller was developed that minimizes the energy use of the HEUAV. During HEV mode, the optimum power split between the engine and the electric motor was determined. The operator in the loop provides the closed-loop feedback to maintain the desired UAV speed and altitude. The operator could be a human at a control station or an autonomous, pre-programmed computer.

The research approach involved four steps. First, an HEUAV model was developed that is based on an HEV model developed by researchers at the HEV Center for Visteon. The model, described in detail in Chapter 4, was updated to include a model for a UAV and propeller instead of an automobile and a transmission. The specific data for the ICE, EM, batteries, and other components were updated to match the appropriate components for the simulated UAV. The model was used to test the control algorithms. Second, a conceptual design approach was used to optimize the size of the wing and the propulsion system components for the small UAV. Third, the UAV model was run with an original propulsion system with either a two-stroke or four-stroke gasoline engine to provide baseline results for the hybrid-electric propulsion system. Fourth, the results for the HEV propulsion system with a rule-based controller were compared to the CMAC ANN controller results.

The neural network controller uses a CMAC ANN for the replacement of the rule-based controller as shown in Figure 1-2. The CMAC controller is implemented in the Simulink model using an S-function to permit the code to be directly portable to a microprocessor. The controller has inputs such as demanded torque, propulsion system rotational speed, battery SOC, and UAV speed. The outputs include hybrid mode (engine and clutch engagement command), engine throttle command, and electric motor torque command.

In addition to comparing performance and energy use results between the rule-based and CMAC controllers, the microprocessor memory requirements were analyzed for the ANN controller. The CMAC greatly saves on memory and computational requirements as compared to having a multi-dimensional look-up table or computing a solution hyper-plane for every instant in time.

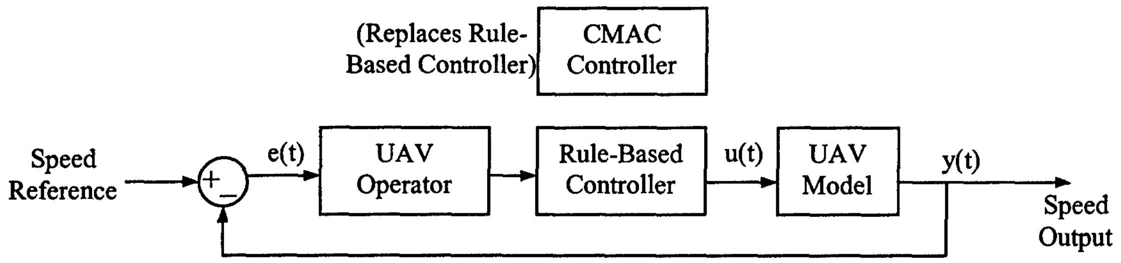


Figure 1-2: HEUAV Power Train Controller

To narrow down the scope of the dissertation research, the research did not include the actual design of the airframe and propulsion system components, cost analysis, or airworthiness analysis. Although a precursory design was completed with off-the-shelf components, it was assumed that each component was designed close to its maximum efficiency. Aircraft balance and flight control system design was also not considered even though any of these items would provide significant and interesting research projects. The research did not involve analyzing the reduction of emissions from the hybrid-electric propulsion system.

1.4 Research Focus Areas

Several problem areas were the focus of the research and include **contributions to the fields of hybrid-electric propulsion system control, unmanned aerial vehicles, and neural network control.**

Advanced Control Algorithm for Electric-Only/Hybrid Operation: To determine the best use of available energy in the hybrid-electric propulsion system, two primary items

The efficiency maps of the ICE, EM, and the batteries and appropriate parameters (demanded torque, rotational speed, SOC) were used to form a "hyper-plane" based on an instantaneous optimization algorithm. This hyper-plane is used to train the CMAC neural network. The algorithm includes a performance measure to minimize the total power consumption. The CMAC neural network controller replaces the rule-based controller and the results are compared.

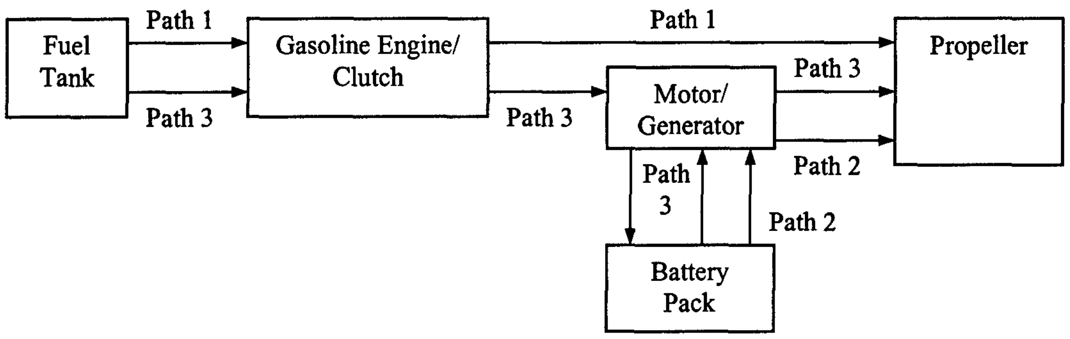


Figure 1-3: Energy Paths for the Parallel Hybrid-Electric Propulsion System

Real-Time, Minimal Computation Control Algorithm: One of the disadvantages of many of the control algorithms such as optimal control is that the solutions must be computed off-line due to the large computational requirements. Rules are then derived from the results and programmed into the microcontroller. As the CMAC neural network was developed for the HEUAV application, the potential microprocessor was considered in order to choose a neural network structure that is consistent with the memory capabilities of a microprocessor. Although the CMAC controller was initially trained off-line, the CMAC's association memory is capable of being stored in the memory of the microprocessor. The computational requirements for the CMAC are minimal which would allow an adaptive scheme to be implemented in follow-on research.

were determined: the times when the vehicle should be in hybrid or electric-only mode and if in hybrid mode, the torque split between the engine and the motor. Control strategies for a parallel HEV configuration usually determine the time to be in hybrid mode based on the speed of the vehicle. The HEUAV took the same approach and normally operates in hybrid mode unless flying at endurance speed where it operates in electric-only mode (stealth mode). While in hybrid mode near cruise speed, power is minimized by attempting to run the engine in its most efficient operating region: the ideal operating line (IOL) or ideal operating band (IOB) [43]. For low torque demand, a large amount of energy can be used for regeneration by using the engine to supply energy to the motor/generator, to the batteries, and then back to the electric motor. The energy conversion losses reduce the overall efficiency of the system. One of the areas of research was to determine when the engine should be idled back and still provide direct mechanical energy to the drive train or when it should be operated on or near the IOL and use the additional energy to recharge the batteries. Computing the amount of energy exchange with the batteries as compared to using the engine in a lower efficiency region is not a straight forward problem due to the nonlinear efficiency maps for the engine and motor and differing charge and recharge battery resistances [44]. As shown in Figure 1-3, the energy available in the gasoline and the batteries can follow one of three paths: directly to the propeller from the fuel tank/engine, directly to the propeller from the battery pack/motor, or indirectly to the propeller from the fuel tank/engine to the motor, batteries, and back through the motor. While in hybrid mode, the CMAC ANN controller determines the torque split to obtain the most efficient use of the on-board energy for a specific mission.

Conceptual Design of a Hybrid-Electric UAV: A conceptual design of a parallel hybrid-electric propulsion system is given for a small UAV designed for intelligence, surveillance, and reconnaissance (ISR) missions. The conceptual design reveals the trade-offs that must be considered to exploit the advantages of the hybrid-electric propulsion system. The resulting hybrid-electric propulsion system is a two-point design that includes an engine primarily sized for cruise and an electric motor and battery pack that are primarily sized for endurance.

1.5 Overview of Dissertation

A CMAC neural network that controls the energy use of a hybrid-electric propulsion system for a small UAV was developed in this research. The parallel HEV expertise at UCD is applied to a UAV with the potential of satisfying military, homeland security, and disaster-monitoring missions. The electric-only operation of the UAV provides stealth operation. The CMAC neural network controller is applied to a small UAV with a fixed-pitch propeller, electric motor, gasoline engine, and lithium-ion battery pack. The major areas of research were to develop the advanced control algorithm to minimize (i.e. optimize) the energy use of all of the propulsion system components and to design a control algorithm that could be used in real-time with minimal computational and memory requirements. The developed control algorithms for the UAV are also applicable to other HEV applications.

Chapter 2 of the dissertation includes the literature search and explains in detail the various control algorithms that others have used for the control of hybrid-electric power trains. Chapter 3 provides an overview of the CMAC neural network. Chapter 4 explains in detail the MATLAB/Simulink model that was developed for the HEUAV.

Although the focus of the research was on the advanced control algorithm, a typical design process was required to select satisfactory and nominal propulsion system components for the HEUAV. The conceptual sizing of the wing and the propulsion system components are explained in Chapter 5. Chapter 6 provides the simulation results of the CMAC controller for typical ISR missions and comparisons to a rule-based controller. Conclusions and recommendations for future research are included in Chapter 7. The Appendix includes the MATLAB and C++ code used for the conceptual design, HEUAV Simulink model, and CMAC artificial neural network.

Chapter 2: Hybrid-Electric Power Train Control Approaches

2.1 Power Train Control of Hybrid-Electric Vehicles

The power train control approaches for a hybrid-electric vehicle (HEV) can be classified into two general categories:

- Rule-Based Strategies Based on Practical Experience or Empirical Data
- Advanced Control Algorithms

Although the first category has resulted in many successful and reliable implementations such as in the ADVISOR HEV simulation software and the FutureTruck competitions, other more advanced control techniques such as optimal control, fuzzy logic, and neural networks can potentially be used to further improve the fuel economy and performance of the power train. The advanced control algorithms build on the experience-based algorithms and are not necessarily a replacement but a supplement to the control approaches based on empirical data or experience.

Several advanced control algorithms have been used for the control of the HEV power train in automotive applications:

- Optimal Control
- Fuzzy Logic Control
- Artificial Neural Networks
- Adaptive Control
- Nonlinear Control
- Genetic Algorithms

A brief overview of the first four and their application to the control of HEV power trains, as found in the literature, is given in this chapter. Optimal control, fuzzy logic, neural networks, and adaptive control are the focus of the overview, although several researchers have applied nonlinear control [45, 46] and genetic algorithms [47] to their HEV application. This overview provided the technical background necessary to select the appropriate or desired control system for the hybrid-electric unmanned aerial vehicle (HEUAV) propulsion system.

2.2 Rule-Based Control Strategies

The rule-based control strategies based on lab data and experience have resulted in successful, reliable, and efficient HEV power trains. These control strategies have been used in the ADVISOR HEV simulation software [44] and the FutureTruck competitions [16, 48]. The approach can vary depending on whether the HEV is a parallel or series configuration or whether the vehicle will be operating with an electric-only, charge-sustaining, or charge-depleting strategy [27]. Other factors include the goals of the designer or researcher, the source of the electricity, type of fuel, size of the vehicle, performance requirements, and the application of the vehicle.

The overall objective for the rule-based or any advanced control algorithm is to optimize the energy use of the power train or to reduce the emissions. For all of the rule-based implementations, an attempt is made to operate the internal combustion engine (ICE) and the electric motor (EM) in their most efficient regions. Since operating one energy source in its most efficient region usually does not permit the other one to operate in its most efficient region, trade-offs must be made. For example, the engine in a parallel HEV is usually not operated at low vehicle speeds where it is very inefficient.

Once the vehicle is at a higher speed, the ICE can be used along with the motor to propel the vehicle. The ICE is used at low vehicle speeds only if the battery state-of-charge (SOC) is low. The efficiency maps of the engine and electric motor, along with the expected energy conversion losses in the batteries and other components, are used to minimize the energy use of the power train.

Several publications focus on the modeling of the power train and give a brief description of the rule-based algorithm. Cikanek and Powell describe in detail their models used for the engine, motor, clutch, transmission, batteries, and the vehicle [49-51]. The models are intended to be used in a plug-and-play environment to test different control strategies. Relatively simple logic rules are used to validate the power train model. Yang also describes in detail a forward-facing HEV power train model that was built in MATLAB/Simulink [52]. The model is based on rigid body motion similar to many of the other models. The power train controller includes traditional proportional-integral-derivative (PID) control with switching logic and outputs control signals to the motor, engine, and transmission. He implements a model based on empirical data for an engine-assisted parallel HEV [53]. Bowles uses relatively simple logic in a charge-sustaining parallel HEV power train model to keep the battery SOC constant [54]. Chang and Buntin also developed a model based on previous work by Powell and others that is used in a logic-based control algorithm [55, 56]. Finally, Bailey uses logic in his supervisory controller to provide seamless transitions from one hybrid mode to another [57]. The very detailed models allow the users to obtain accurate simulation results and provide the ability to test multiple control algorithms.

For series HEVs, three control strategies based on logic or rules are found in the literature. First, the thermostat method turns the auxiliary power unit (APU) on and off based on the battery SOC. The minimum and maximum SOC values are determined by examining the efficiency maps of the battery pack [58, 59]. The battery pack is operated in the most efficient region. Second, the load-following method controls the output of the APU based on the rate of change of the battery SOC. Third, the power split strategy usually used in parallel hybrids is utilized by Jalil for a series HEV to improve the fuel economy over the thermostat method [58]. The power split control logic in the series hybrid determines if the energy for the electric motor comes from the APU directly or if it comes from the batteries. The thermostat, load-following, and power split strategies are all effective control strategies for series HEVs.

The parallel plug-in HEVs designed by the HEV Center at UCD use logic based on experience, simulations, and intuition to produce effective control strategies [16, 24, 60]. The plug-in vehicles have relatively large battery packs to maximize the all-electric range. The vehicles primarily operate in a charge-depleting strategy to also maximize the off-board electricity use. If the battery SOC drops to a low level, the power train enters a charge-sustaining hybrid mode. The overall approach is to operate in electric-only mode at low vehicle speeds such as in the city where the ICE is inefficient and to operate in hybrid mode at highway speeds or when the battery SOC is low [61]. The ICE is operated on or near the ideal operating line for the best efficiency during hybrid mode [43, 62]. The electric motor provides additional torque or acts as a generator during regenerative braking. While in hybrid mode, power split strategies are used to minimize the overall energy use or to minimize emissions. The rule-based control strategies seen in the literature based on lab data, simulations, and experience have resulted in successful and reliable HEV power trains but other more advanced control algorithms have the potential of improving the fuel economy or further reducing the emissions.

2.3 Optimal Control

The objective of optimal control is to "determine control signals that will cause a process (plant) to satisfy some physical constraints and at the same time extremize (maximize or minimize) a chosen performance criterion" [63]. The plant is usually described in state variable form and the physical constraints are applied to the state variables. The performance criterion or cost function can be formulated either to minimize transient energy, control energy, fuel usage, or time [63, 64]. For an HEV power train, the fundamental objective is to either improve the fuel economy or reduce emissions. The optimal control problem then is to find the optimal power split between the engine and the electric motor in hybrid mode, or the optimal energy use between gas and electricity to obtain the desired speed on the driving cycle. Other parameters such as the SOC of the battery and the gear ratio of a continuously variable transmission (CVT) may be used to determine the optimal solution. The optimal solution can be determined for an instantaneous optimization or for a global optimization such as over an entire driving cycle. Although the concept of optimal control is intuitive, implementation for an HEV power train, especially for global optimization, is much more difficult. Various optimal control schemes that have been applied to the power train control of an HEV will be discussed along with the different computational algorithms required to implement the control schemes.

The most difficult aspects of implementing an optimal control scheme for the HEV application are to determine an accurate model and cost function, to estimate the final conditions if a global optimization is desired, and to use a computational algorithm that can be computed in real-time. The HEV power train model can either be nonlinear or approximated as a set of linear functions. Since an HEV power train includes highly nonlinear components, it can be difficult to obtain an accurate linear model that captures the dynamics of the power train. An accurate cost function can also be difficult to determine because of the nonlinear components. In addition to the challenge of deriving an accurate power train model and cost function, the classic solution to the optimal control problem requires initial conditions for the state equation and final conditions for the costate equation (adjoint system). The cost function, state, and costate equations are used to determine the optimal control inputs for the desired time interval. Final conditions (or states) for a global optimization scheme are often difficult to determine for an algorithm that needs to be computed real-time in an embedded microcontroller. The driving route is usually not known *a priori;* therefore, it is difficult to select the final states of the system. The solution for the fundamental optimal control problem is backward-in-time and is very effective for applications such as pre-planned aircraft routes or a known path such as for a bicycle or auto race. Even if the final conditions can be determined, the computational algorithms required usually are difficult to implement in an embedded microcontroller. Most research papers state that the optimal control solution is for simulation only and that more work needs to be done on algorithms that can be used in real-time. The limitations are due to the computational intensive algorithms and the look-ahead requirements for most global optimal control implementations. The challenge of applying optimal control to the control of an HEV power train is to find a realistic, concise, linear or nonlinear model that accurately represents the drive train, determine an appropriate cost function, and to implement the solution in a real-time embedded microcontroller.

The optimal control implementations seen in the literature are either global or instantaneous solutions. The global implementations are intended more for simulations due to the required look-ahead final conditions and the computational requirements. The instantaneous optimization implementations solve the optimal solution during each time step and not over the entire driving cycle [27, 60, 65, 66]. The global optimization schemes generally achieve higher fuel economy but cannot be implemented effectively in real-time.

Optimal control for instantaneous optimization has been used in simulations by Kleimaier and Kim to improve the fuel economy of a parallel HEV with a CVT [14, 48]. Kleimaier's objective is to minimize the drive train's instantaneous power dissipation. The controller uses the cost function:

$$J = P_{Diss} + \alpha \cdot P_{Batt} + K \tag{2-1}$$

where P_{Diss} is the power dissipated by the engine, motor, and CVT, P_{Batt} is the change in the stored battery energy, α is the weighting factor for the electric energy use, and K primarily determines either hybrid or electric mode. Electric energy is increased for city driving by decreasing α and minimizing the penalty for electricity use. For highway driving, the use of electric energy can be penalized by increasing α. With the implementation of the cost function, concerns such as the drive train components' range of operation and the SOC of the battery pack have to be considered resulting in a constrained optimal control problem. The plot of the cost function has local minima that the controller uses to give engine and motor controller commands to minimize the power consumption of the drive train. The battery SOC is explicitly used in the cost function and not as a constraint since the cost function is intended to be used in real-time. The scheme is an instantaneous optimization one, but parameters such as α are determined in off-line simulations. Kleimaier is currently attempting to more efficiently implement the code so it can be operated in real-time. Kim takes a similar approach in a simulation and defines a cost function related to the effective specific fuel consumption defined as the fuel burned per output kWh [14]. An optimum operation line is used to determine the control variables such as the CVT gear ratio, motor torque, and engine throttle as a function of vehicle speed, battery SOC, and required power. The state variables include the engine and motor torque, motor speed, the battery SOC, and the power required. The battery and electric motor are transformed into an equivalent ICE model and the hybrid system is modeled as two engines. The optimal operation line is found by repeatedly calculating the specific fuel consumption for all allowable combinations of the system state variables. Kleimaier and Kim both effectively use the optimal control approach in their simulations to improve the fuel economy of their parallel HEVs with a CVT, but the challenge is still to implement the algorithms in an actual vehicle controller.

Delprat and Steinmauer have implemented optimal control algorithms in their HEV power train simulations that are intended to be implemented in real-time. Delprat uses a global optimal control algorithm in a simulation of a parallel HEV power train with a two-speed transmission [67, 68]. The approach uses the battery SOC as a state variable, and the overall goal is to take the battery SOC from an initial state to a final state and minimize the cost function, J, that primarily involves the fuel consumption. The constrained optimal control formulation takes into consideration the torque and speed limits of the engine and motor and the inefficiencies of the components. To find the minimum of the function, partial derivatives and Lagrange multipliers are used for either negative or positive electric motor torque. The resulting optimal control formulation is relatively computationally efficient and is intended to be used real-time and not just for simulations. Another similar approach by Steinmauer in a simulation also uses the fuel consumption in the cost function, and the battery current is calculated for each time step to minimize the cost function [69]. This determines the times when the battery power is used or when the engine/generator is turned on in the series HEV. A simplex algorithm is used that results in minimum fuel consumption. Both of these optimal control approaches may eventually be implemented in a real-time controller.

Optimal control can effectively be used with nonlinear models [70]. The linear models of the highly nonlinear drive train components of an HEV are often unable to accurately predict the actual performance of the system. Lyshevski developed nonlinear models of a series-hybrid diesel-electric power train and then used these models in an optimal control scheme [71, 72]. The nonlinear controller is designed using Lyapunov-based theory and is a constrained controller due to the bounded control inputs such as the injected fuel and voltages applied to the field windings of the electric motor. The theory for the constrained optimal controller using nonlinear models is based on the theory developed by Lyshevski [73]. The constrained optimization problem with nonquadratic functionals is solved using dynamic programming. The optimization approach extends the Hamilton-Jacobi theory using nonquadratic functionals [74]. The constrained optimal controller using nonlinear models and adaptive programming could be an effective approach for the control system of an HEV power train.

The optimal control formulation is used by some researchers as one of the tools in more complicated control schemes for HEV power trains. For example, Saeks uses a linear optimal controller as a part of an adaptive control scheme for a four-wheel drive HEV [75]. Zhang uses an optimal controller as a component of an HEV controller based on hybrid dynamical systems theory [76]. The HEV is modeled as a complex hybrid system with numerous subsystems. The information flow is via signals that are either discrete or continuous. The hierarchical structure includes supervision, coordination, and execution levels. The subsystem-specific controllers are implemented at the execution level and the optimal controller is implemented at the coordination level. Dynamic programming is not used to solve the optimal controller problem due to the large memory requirement. The nonlinear optimal control problem is simplified and a sequential quadratic programming (SQP) technique is used to solve the second order approximation for the cost function and constraints. The literature reveals that optimal control can be effectively used as a tool in more complicated control architectures.

The optimization problem, especially for global optimization, associated with HEV power train models and cost functions is usually too complex to be solved analytically; therefore, a numerical solution is required. Techniques used to solve a dynamic optimal control problem include calculus of variations, the Pontryagin principle, and dynamic programming. Numerical techniques such as Runge-Kutta are not applicable due to the split boundary conditions [77]. As Lyshevski states, "[o]ne of the major problems in optimal control is the lack of efficient methods which allow one to solve constrained control problems for high-order systems" [73]. Most of the methods use gradients that require the calculation of the Hamiltonian and the adjoint (costate) differential equations [78]. The nonlinear HEV models and lack of smooth surfaces for the cost function make the derivative-required algorithms difficult to use. A method that does not require the use of the Hamiltonian is the sequential quadratic programming (SQP) method. A SQP method is used by Zhang, Kleimaier, and Zoelch to solve their optimization problems [76, 78, 79]. The SQP is currently only a simulation technique since the memory requirements are too large for an embedded real-time controller. Most derivative-free optimization techniques such as direct rectangles (DIRECT) and simulated annealing (SA) usually require a large number of iterations which make them unusable for real-time application [80]. These techniques are useful to determine the optimal solution and can be used to formulate a rule-based strategy that closely mimics the results of the off-line simulation.

Dynamic programming techniques can be useful in the solution of the global energy optimization problem, but they are difficult to implement in real-time [81, 82]. Lin applies dynamic programming to solve the minimal fuel optimal control problem for a charge-sustaining parallel hybrid-electric truck [83]. A dynamic optimal solution over a complete driving cycle is developed but cannot be implemented on the vehicle due to the look-ahead requirement of the algorithm and because of the heavy computational requirement. The cost function to minimize the fuel consumption takes the following form:

$$J = \sum_{k=0}^{N-1} L(x(k), u(k)) \tag{2-2}$$

where N is the length of the driving cycle and L is the fuel consumption rate. U(k) is the vector of control variables such as the fuel injection rate and output torque and x(k) is the state vector. Since the application is a constrained optimization scheme, constraints are applied to the engine and motor speeds, motor torque, battery SOC, and other parameters. The optimization problem is solved using dynamic programming based on Bellman's principle of optimality [84]. The disadvantage of this approach is the requirement to solve the recursive equation backwards from time N-1. Therefore, simple rules were derived from the optimal control simulations using neural networks and implemented in the vehicle. Brahma also uses dynamic programming to determine the optimum power split for either electrical energy generation or storage for a series HEV [85, 86]. The optimal solutions again are based on known driving cycles so a load forecasting algorithm would be needed to apply this method to a real-time application.

Saeks uses adaptive dynamic programming (ADP) techniques to solve for the optimal control law in real-time. He states that adaptive dynamic programming:

> methods were introduced into neural control in the past decade to circumvent the classical conundrum of optimal control theory where "the optimal cost (or return) function is required to compute the optimal control law and vice versa." The Calculus of Variations attempts to circumvent this problem by directly evaluating the optimal control law while dynamic programming takes the opposite approach by solving Bellman's equation for the optimal cost function. ADP methods, on the other hand, are designed to learn the optimal cost function and the optimal control law on-line, converging asymptotically to the optimal solution. At each time step,

the controller estimates the optimal cost of taking the system from the present state to the final state, and uses that estimate to specify a control law at that time step, while simultaneously updating the estimate [75].

ADP methods could potentially solve for the optimal solution in an embedded controller for an HEV application.

A method related to optimal control was used by Morchin to minimize the energy use of a hybrid-electric bicycle [87]. The equations for the energy use of the bicycle due to acceleration, friction, air drag, and hill climbing are combined, and Lagrange's calculus is used to find minimums in the two-dimensional surface [88]. The disadvantage of this technique is that the speed and driving cycle must be known beforehand to compute the optimum energy use. The technique is useful for races or known commuting routes so the optimal energy use can be computed, but it is not as beneficial for unknown riding paths or commuting routes.

Optimal control could be used for the control of the HEUAV propulsion system. If a realistic linear model can be developed along with a quadratic cost function, then the traditional optimal control and algorithms can be applied. If a nonlinear model or a nonquadratic cost function is needed, the theory and solution techniques are not as well developed. The other possible disadvantage is the apparent lack of computational algorithms that can be implemented in real-time. Furthermore, the classical solution techniques require final conditions that may be difficult to determine. Based on the literature, the instantaneous optimization schemes have been more successful than the global optimal control implementations, even though global optimization schemes provide better fuel economy.

2.4 Fuzzy Logic Control

Fuzzy logic controllers (FLC) use linguistic knowledge that contains the intuition and experience of an operator or designer [89]. Fuzzy control is described by Jantzen as "control with sentences rather than equations" [90]. Fuzzy control can be used in a wide range of applications, but it is often used to handle high-level, supervisory control functions that original control theory has difficulty in addressing [91]. A brief overview of fuzzy logic will be given and how it has been used in HEV power train controllers.

Fuzzy logic (FL) originated with a publication written by Zadeh entitled "Fuzzy Sets" [92]. The theory, an extension of multi-valued logic, was termed "fuzzy logic" as compared to binary, crisp, or Boolean logic [93]. In traditional binary set theory where parameters can only take on values of 0 or 1, parameters in fuzzy set theory can take on values anywhere between 0 and 1 [94]. The binary values are at the extremes of the fuzzy range. The fuzziness of the data is modeled after human thought and classification processes where there is uncertainty in the accuracy or the confidence of the data [95]. More specifically, the elements in a fuzzy set have a degree of membership dependent on the membership or characteristic functions that can be triangular, trapezoidal, or other shapes. The membership functions determine the actual fuzzy values in the system. A fuzzy variable has values that are labels of fuzzy sets and usually take on linguistic values such as "small," "medium," and "large."

Fuzzy variables can be used in conditional statements to generate fuzzy rules that mimic human thought. The conditional statements are similar to those found in software programming using terms such as IF-THEN. An example of a conditional statement for an HEV application is "IF (Vehicle Speed Low) THEN (ICE Torque is Zero AND EM

Torque is Medium)". The input conditions are referred to as antecedents and the output conditions are called consequents. The use of linguistic rules is a powerful method that can be used in a controller.

The actual FLC controller can be thought of as a black box with inputs and outputs. The FLC uses fuzzy sets, fuzzy rules, and fuzzy logic to determine the outputs from the inputs. The key components of the controller are:

- Rule Base: Fuzzy rules based on knowledge that describe how the FLC performs. The rules form the heart of the controller since the output is based on these rules.
- Fuzzification Interface: Converts the crisp numerical data inputs into fuzzy data.
- Inference Mechanism: Applies the fuzzy rules to the fuzzy input data to determine the linguistic fuzzy outputs. The minimum (Mamdani), max-product, or max-min (Zadeh) implication rules are often used to determine the degree of membership each rule has in the output.
- Defuzzification Interface: Converts the fuzzy outputs to crisp numerical outputs. The center of gravity and center of area methods are often used.

The FLC components permit linguistic knowledge to be embedded in the controller. A complete overview of FLC and the components can be found in several texts [77, 96].

The FLC does not require an accurate model of the system. Fuzzy controllers are useful for complex, nonlinear, multi-dimensional systems, systems with parameter variations, and when the model is unknown [97]. A FLC is reportedly a good choice for HEV applications due to its robustness and tolerance for imprecise measurements and component variability [98]. Several researchers have used FLCs in HEV applications.

Several authors, Salman, Farrall, Lee, Schouten, and Won, have used fuzzy logic in the design of controllers for parallel HEVs. Salman uses fuzzy logic for the energy management and control system for a charge-sustaining parallel HEV [98]. The logic rules are intended to optimize the efficiency of the entire system, not just one component such as the engine or motor. The fuzzy logic rules are based on the driver's inputs, the battery SOC, and the speed of the motor/generator [99]. The fuzzy logic rules generate commands for the diesel ICE, motor, battery, and transmission (5-speed manual). The fuzzy logic controller is a Sugeno-Takagi type that requires membership functions for the inputs but not for the outputs. During the design of the controller, valuable insights were gained. An efficiency map of the batteries revealed that the battery pack is the most efficient at high SOC and low power levels. The ICE and motor/generator are used to charge the batteries only when the ICE is operating close to its highest efficiency. The electric motor propels the vehicle when the efficiency is 16% better than using the ICE directly. This was determined based on the energy conversion losses while charging and discharging the batteries. The resulting strategy only uses the motor at low speeds and power levels, the engine is used during moderate power demands, and the hybrid mode is used at high power demands. The controller was implemented in the forward-looking simulation program PSAT and produced higher fuel economy than the default controller. Farrall also uses fuzzy logic in a controller for a parallel HEV with a five-speed manual transmission [100]. Farrall stresses that the controller is not a controller in the traditional sense since the controller does not actually track a reference input. The driver of the forward-facing model implements the feedback for the entire vehicle. The controller takes the power request from the driver and determines the throttle angle of the engine and the armature current of the electric motor. The logic closely mimics that of a human to determine several outputs based on an input. The models of the vehicle do not include states and there is no cost function as in optimal control. The fuzzy logic controller uses the max-product method for implication. The fuzzy set for the engine was highly nonlinear in order to have high sensitivity in the region where the engine is usually operated. Farrall ends his analysis by stating that the best use of fuzzy logic would include provisions for the decision rules to adapt to the driving conditions. Lee uses dynamometer test results to build the fuzzy logic rules for a charge-sustaining parallel hybrid-electric bus [101, 102]. Max-min composition and the center of gravity method are used in the inference engine and the defuzzification step, respectively. There are two inputs, accelerator pedal position and the electric motor speed, and one output in the controller. The output is the ratio of torque command to the rated torque at a specific speed. The proposed strategy is insensitive to disturbances such as the driver's driving habits, driving route, and the load on the vehicle. Schouten uses a Sugeno-Takagi FLC like Salman to either optimize the fuel economy or to minimize emissions [103]. Membership functions are derived for the driver input, SOC, electric motor speed, gear ratio, and the vehicle speed. Output commands are generated for the ICE, EM, and mechanical braking. The rule base is split up based on the desired goal of the driver: performance, fuel economy, or braking. Won implements a parallel FLC in a hierarchal structure that incorporates modes such as start, acceleration, deceleration, and cruise [104]. A specific set of rules is developed for each mode. Different energy management strategies can be used such as charge-sustaining or electric only. All of the fuzzy logic controllers were successful designs for the parallel HEV applications.

Cerroto and Zahran use fuzzy logic in the controllers for their series HEVs. Cerroto uses a FLC for the energy management in a series hybrid-electric bus [105]. The inputs to the controller are the error of the requested traction power, difference between the engine power and average requested power, and the battery SOC. The outputs of the controller are the power gradient for the generator and the delta between the required drive train power and the power of the generator. The membership functions are generated using exponential laws, and the height method is used for defuzzification. Zahran also applies fuzzy logic control to a series HEV [89] that uses a photovoltaic battery. The author selected fuzzy logic based on the highly nonlinear system and environment. The inputs to the controller are the battery SOC and the battery current. The output of the controller is the change in the current and the logic controls for the diesel engine/generator. The FLC is implemented in a microcontroller. The results indicate that FL can be used in a series HEV in addition to parallel HEV applications.

Several patents have been issued for HEV power train controllers that are based on fuzzy logic. Sakai was issued a patent for the FLC of either a series or parallel charge-sustaining HEV [106]. The fuzzy logic rules use the SOC and the change in the SOC to determine the output commands. Ibaraki received a patent for a FLC designed to control the electric motor in a parallel HEV [107]. The literature reveals that the auto manufacturers have a high interest in the use of FLC for HEV applications.

Fuzzy modeling has been used as a part of an intelligent control scheme by Quigley to predict the duration and distance of a driving route [108, 109]. Data was collected from several vehicles and a subtractive clustering method was used to generate membership functions and rules in a fuzzy inference system [110]. Due to the wide variation of data, the rules developed from the data still must be used in conjunction with data about the energy consumption. This would permit the ICE to be shut off in the latter part of the driving trip when the confidence is high that the driver is almost to the work site, home, or other location. The intelligent control system was able to predict the trips during the week much more accurately than weekend trips due to more repeatable trips during the week.

FL can be used in forward-facing simulations and can eventually be programmed into microcontrollers. The resulting controllers are not computationally intensive and can be used real-time in the vehicle [111]. Koo implemented his FLC in a digital signal processor [101]. The ability to implement a FLC in a real-time controller makes FL a promising method for HEV applications.

Adaptive control techniques can be used to adjust the fuzzy logic rules and the membership functions in the FLC, or the FLC can be used as the adjustment mechanism in the adaptive controller. An adaptive FLC has been used by Rajagopalan in a predictive intelligent control scheme that uses traffic and road conditions [112]. Neural network algorithms or other techniques can be used to update parameters in the FLC [94]. Combining adaptive control, fuzzy logic, and neural networks can potentially provide an effective controller that uses the advantages of each. For other applications, the fuzzy logic supervisory controller can be used to adjust the gains in a PID controller or parameters in other types of controllers [91].

To conclude the fuzzy logic section, FLCs appear to have the largest advantage when used in applications that require high-level, supervisory, task-oriented control instead of regulation where PID control is sufficient. FLC permits the designer to articulate thoughts, experience, and intuition into a powerful controller that mimics the preferences of the designer. Fuzzy logic can be used with other control strategies to take advantage of the benefits of each.

2.5 Artificial Neural Networks

Artificial neural networks (ANN) can be used to estimate a plant model or replace a controller in adaptive control schemes. An ANN is intended to learn in a similar manner as the human brain. Researchers study biological nervous systems with the purpose of designing computational systems that have brain-like capabilities. The artificial neurons of a neural network are arranged to process information in parallel and to send/receive information from other neurons similar to biological systems. Neural networks are receiving a great deal of attention in the current literature and are being used in many applications such as image and voice recognition, system identification, and control. As with any control algorithm, neural networks have advantages and disadvantages. Neural networks are useful in the control of nonlinear multi-variable plants, are capable of learning from a training set, and parallel processing is inherent. The disadvantages include the difficulty of extracting the knowledge base contained in the net, predicting results for cases outside the training set, and the convergence and training time. Several excellent surveys of neural network control can be found in the papers by Agarwal [113], Hunt [114], and Narendra [115]. Some of the recent work using neural networks for optimization problems may be valuable for an HEV application.

The concept of ANNs originated with McCulloch and Pitts and the perceptron was later introduced by Rosenblatt [96]. Despite research in the area, ANNs were not fully understood until the 1980s as a result of the research by Hopfield and others [116]. The most commonly used neural network in applications is the Multi-Layer Perceptron (MLP) network which is a direct extension of the networks using perceptrons and adaptive linear neurons (ADALINE) proposed years ago. The interest and research in the area of ANNs has exploded since the 1980s, and the conference and journal papers are too numerous to list.

The basic neuron model includes inputs, weights, a summation, an activation function, and an output as shown in Figure 2-1 [117]. The inputs can come from other neurons or external inputs and are multiplied by adjustable weights corresponding to biological synapses. The weights are determined by using a training algorithm. The weighted inputs are summed, and an activation function determines the output of the neuron. The most common activation functions are linear (ADALINE), binary, sigmoid, hyperbolic tangent, or perceptrons [96]. The output of the neuron varies between zero and one. The simple neuron model is used as a building block for the more complex nonlinear ANN.

The architecture or topology of an ANN can be classified into the following: Hopfield recurrent network, feed-forward network, feedback networks, and symmetric auto-associative networks [96]. The Hopfield recurrent network has nodes which can interact with nodes in the same, higher, or lower layers. The information in a feed-forward network only flows from the lower (input) to higher (output) layers. In the feedback network, the information flows from the output to the input. The last topology has connections and weights that are symmetric. Most topologies are multi-layered and use a hidden layer of neurons between the input and output layers. The Hopfield ANN topology is often used in control applications due to its advantages that it can learn from experience rather than programming, can generalize, can generate nonlinear mappings, and is a powerful parallel processing setup. ANNs, in general, are capable of producing any arbitrary input-output mapping which is critical for nonlinear systems. The application will often determine the best type of ANN to use.

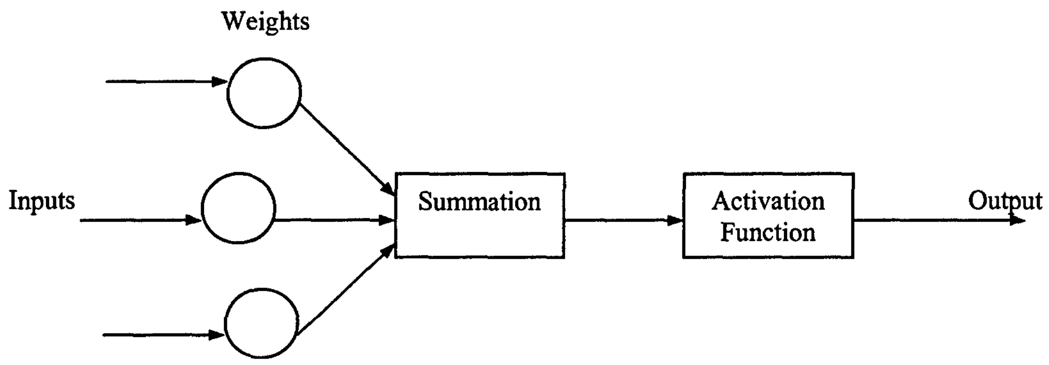


Figure 2-1: Neuron Model

One of several algorithms is used to train an ANN and is an active area of research. The two basic classes of ANN training are supervised and unsupervised. Supervised learning requires an external source for knowledge or data as compared to unsupervised training that requires no external source of information. More specifically, supervised training involves a structure that produces an output that corresponds to the input. The popular back-propagation algorithm is a member of the supervised training class. For unsupervised training algorithms (open-loop adaptation), no information exists for the desired output and how it corresponds to the input. The unsupervised training algorithms are often used for associative memories and classification algorithms. The associative memories generally have fast initial learning and convergence. Associative memory networks have features which make them useful for on-line modeling and control applications.

Traditional training algorithms determine the weights in an ANN. Once trained with operational data, the ANN can extrapolate to make decisions. Many of the training algorithms are based on traditional gradient-based algorithms. There is a close relationship between the necessary conditions of optimal control theory and the gradient algorithms for training an ANN.

Neural networks can be used for system identification as a part of an indirect adaptive control scheme, and examples are seen in the literature. ANNs can be used as the adjustment mechanism to update parameters in direct adaptive control schemes. Adaptive control architectures are discussed in more detail later.

The use of neural networks for HEV applications in the literature has been limited. Yuan uses a neural network for a series HEV controller [118]. Two ANNs are used in an indirect adaptive control scheme: one ANN is used for system ID and the other ANN is used in the power train controller. ANNs have been used by Swan, Baumann, and Bhatikar for battery modeling [119-121].

The lack of ANNs in the academic environment for HEV applications opens up a door for potential contributions. The work by Hopfield, Tank, Tagliarini and others illustrate the usefulness of ANNs for optimization problems and the ability to apply them in real-time controllers [122]. The fundamental concept for applying ANNs to optimization problems is the fact that recurrent (feedback) networks stabilize and the stabilized ANN and the associated states can be directly related to local minima of an optimization problem. If the ANN satisfies specific stability criterion developed by Cohen-Grossberg, then the ANN will converge. Additionally, the converged, stable solution can be related to the solution of the optimization problem [117]. If the stability criteria exist, a Lyapunov function can be found. A Lyapunov function places constraints on the equations describing the system [117]. Hopfield referred to the Lyapunov function for his specific ANN as an "energy function" that can correspond to the energy in a physical system. Hopfield and Tank initially had the insight to use an ANN that seeks to minimize the energy function and to associate variables in an optimization problem to variables in the ANN's energy function [122]. This insight could prove invaluable for an embedded controller in an HEV application.

2.6 Adaptive Control

An adaptive controller is "a controller with adjustable parameters and a mechanism for adjusting the parameters" [84]. The architecture of an adaptive control system has a control loop and an adjustment mechanism loop. An adaptive controller is nonlinear; therefore, it is more complicated than a PID controller. In this overview, two types of adaptive controllers and the applications that can benefit from the use of an adaptive controller are discussed. A discussion of several journal papers describing the application of adaptive control to the energy management of an HEV is also included.

Adaptive control methods are roughly categorized as either direct or indirect. In a direct adaptive control method, an explicit model of the controller is created, and the controller parameters are directly changed by the adjustment mechanism. The model-reference adaptive system (MRAS) is one form of the direct method as shown in Figure 2-2 [84]. The performance specifications are given in terms of a reference model which is usually a linear, low-order dynamic model. The adaptive mechanism adjusts the controller parameters so that the error between the output of the plant, y(t), and the output of the reference model, $y_m(t)$, is small. The most challenging aspects of the MRAS are to select a suitable reference model and to determine an adjustment mechanism that is stable and also brings the error to zero.

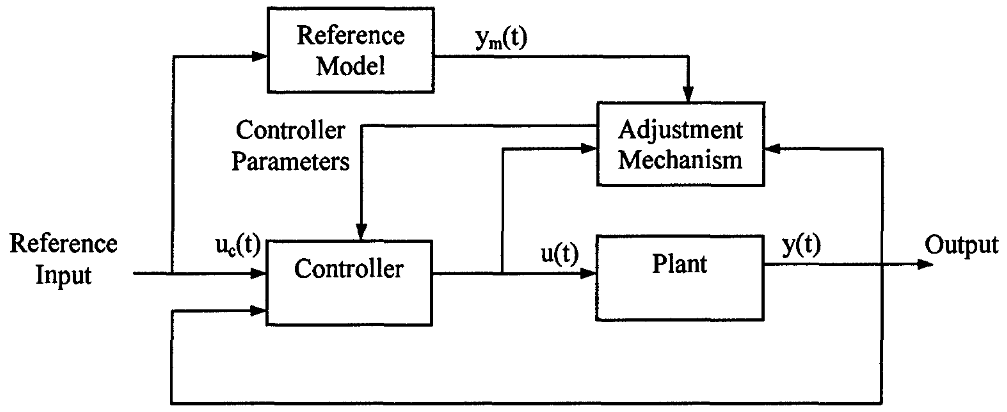


Figure 2-2: Block Diagram of a Direct Adaptive Controller

An indirect adaptive control method produces an estimate of the plant and an inverse method generates the control law. A specific formulation of the indirect method is the self-tuning regulator (STR) as shown in Figure 2-3. The STR obtains estimates of the plant parameters, and a controller design algorithm changes the controller parameters using the estimated parameters. The controller parameters are updated indirectly by the controller design algorithm. The STR approach is very flexible since numerous techniques can be used for the estimator and the controller design algorithm. The estimator and controller can be designed separately due to the separation principle [46].

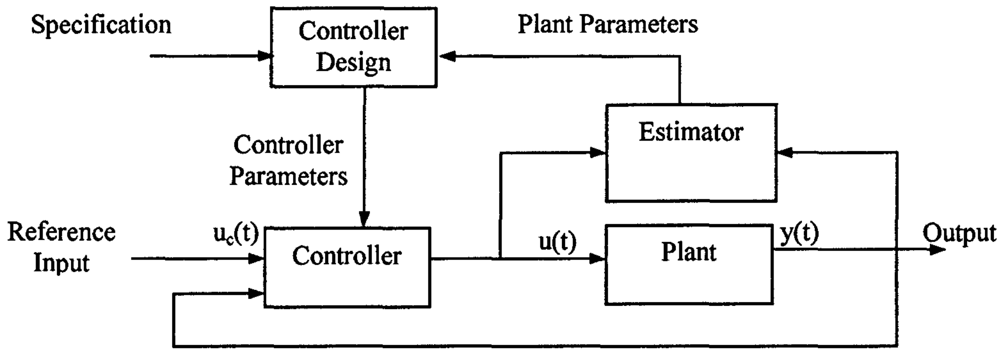


Figure 2-3: Block Diagram of an Indirect Adaptive Controller

Adaptive control methods are beneficial for some specific applications. Adaptive control is useful when the plant dynamics change, if the characteristics of the disturbances change, and to improve engineering efficiency [84]. The plant dynamics may be unknown originally, or if they are known, may change once the system has been operating for a period of time such as those in power systems [46]. Without the use of adaptive control, the system may become unstable due to the changing or unknown plant dynamics. Industrial applications that benefit from adaptive control are nonlinear actuators, robots, steering, aircraft control, and process control. The benefits of using adaptive control vs. a more simplified approach must be determined for a specific application. Many times a constant feedback control system can perform as well as an adaptive controller.

Saeks uses a decentralized adaptive control system for a four-wheel drive HEV [75, 123]. The four motor-generator four-wheel drive HEV is intended for heavier vehicles such as trucks and buses. The HEV uses a fuel cell for the energy converter and a flywheel for energy storage. Four separate but connected adaptive controllers are used to control the vehicle's speed, steering, side slip, and energy management. For the energy management controller, a neural adaptive controller and a linear adaptive dynamic programming (ADP) controller are cascaded to facilitate the use of a linear optimal controller that is at the heart of the energy-management controller. The linear cost function for the system involves the motor speed error, ω_m, and fuel cell power, P_c, subject to the constraint that the flywheel power is finite as shown below:

$$J = \int_0^\infty \left[\alpha \cdot (\omega_m)^2 + \beta \cdot P_c\right] \cdot d\tau \qquad \textbf{(2-3)}$$

Linear models of the motor, fuel cell, and flywheel are used in the state model. Since the model of the vehicle is highly nonlinear, the neural adaptive controller minimizes the speed error and produces a motor speed command. The linear adaptive dynamic programming controller determines the fuel cell, flywheel, and motor-generator commands. The state variables include the electric motor speed and current, the fuel cell current, and the fuel consumption. The control input variables include the voltage of the electric motor, the fuel cell voltage, and the commanded motor speed. The Lagrange multiplier theorem is used to transform the constrained optimization problem into an unconstrained optimization problem. The adaptive controller is compared to a PID controller. At specific design points, the PID performed as well as the advanced adaptive controller, but the adaptive controller has a larger dynamic range.

Several patents for the control of an HEV based on adaptive control have been issued. Drozdz uses an adaptive control scheme for the energy management of a series or parallel HEV [28]. The performance of the HEV power train is analyzed on-board the vehicle in real-time, and the control algorithm changes based on performance data. Schmitz uses an adaptive scheme based on the battery temperature and SOC, regenerative braking mode, and the operating state of the electric motor [124]. The patents illustrate that adaptive control schemes can effectively be used for the energy management of an HEV.

Adaptive control for the hybrid-electric power train of an HEV has been used by several researchers. For one application, the adaptive control algorithm in the energy management scheme permits a more simplified optimal control solution to be found.

Others have attempted to modify control laws based on the driving conditions or driver requests.

2.7 Advanced Control Algorithm Summary

Listed in Table 2-1 is a comparison of the different control algorithms used for the control of HEV power trains. As with any engineering application, control approaches each have their advantages and disadvantages and are listed in the table. The specific concerns for the HEV application are also listed in the table. The control approaches can be combined in numerous ways to take advantage of the benefits of each as in neuro-fuzzy controllers.

Table 2-1: Advanced Control Algorithms for HEV Power Train Control

Control Algorithm	**Advantages**	**Disadvantages**	**Comments**
Optimal Control	1. HEV power train application is an optimization problem (fuel economy or emissions). 2. Well-developed theory for linear models and quadratic cost functionals.	1. Theory not as well developed for nonlinear models and nonquadratic functionals. 2. Classical solution requires final conditions that may be difficult to determine. 3. Computational algorithms are usually not implementable in real-time. 4. Accurate model needed.	1. Global optimization implementations provide better results than instantaneous optimization implementations but are more difficult to implement.
Fuzzy Logic	1. Controller design based on intuition, experience, or knowledge. 2. Useful for nonlinear or complicated multi-variable mechatronic systems. 3. Requires minimal computing power. 4. Accurate model not needed. 5. Uncertainty and fault tolerant.	1. Lack of robustness and stability theory.	1. FLC useful for high-level supervisory control such as a power train controller where components have their own controller. 2. FLC can be very powerful if an adaptive mechanism updates the rules.
Artificial Neural Networks	1. Ability to approximate nonlinear functions or complicated multi-variable systems. 2. Parallel processing that permits high computational speeds and real-time implementation. 3. Plant model not needed. 4. Reliable and fault tolerant. 5. Inherently adaptable to conditions, can learn, and generalize.	1. Difficult to extract the knowledge base. 2. Convergence and training time may be long for some neural networks. 3. Usually trained from operational data. 4. Lack of stability and convergence theory.	1. Can be used for nonlinear system ID. 2. Computational efficiency allows on-line learning. 3. Learning algorithms often based on optimal control techniques. 4. Can be applied to optimization problems. 5. For associative memory networks, rules may be extracted.
Adaptive Control	1. Useful when plant dynamics significantly change or are unknown. 2. Useful when the characteristics of the disturbances change.	1. Existing adaptive techniques usually require a linear parameterization of the nonlinear plant dynamics.	

BLANK

Chapter 3: CMAC Artificial Neural Network

3.1 Introduction to the CMAC ANN

The goal of the advanced control system is to use a minimal amount of energy. This is accomplished by finding the best combination of engine torque and motor torque as a function of parameters such as propulsion system rotational speed, torque demand, battery state-of-charge (SOC), and other parameters. Of the potential hybrid-electric vehicle (HEV) advanced control schemes, those based on artificial neural networks (ANNs) or fuzzy logic appear to be the most promising due to the relatively low computational resources needed. Additionally, in many applications, an accurate mathematical model is not needed (an accurate model is needed for simulations). These approaches are also useful for nonlinear and multi-variable systems, can learn, and generalize. While results from many of the other control methods such as optimal control are very good, the computational requirements are too excessive for the embedded microcontrollers and the majority of the theory is for linear models. This chapter describes the neural network control algorithm that manages the energy use of the hybrid-electric unmanned aerial vehicle (HEUAV) propulsion system.

The use of ANNs in HEV applications has been limited, and due to the potential benefits, this particular control method was selected for the HEUAV application. It is well known that ANNs can approximate nonlinear functions, so an ANN has tremendous potential if applied to the control of the nonlinear HEUAV propulsion system. A specific type of neural network, the Cerebellar Model Arithmetic Computer or Cerebellar Model Articulation Controller (CMAC), was chosen for this application due to its rapid training time, practical hardware implementation, and low computational cost [29]. The CMAC is a feedforward, supervised ANN and is an alternative to the more common back-propagation multi-layer perceptron (MLP) network [30]. The CMAC was originally developed to adaptively control robots since it can handle large input spaces, adapt, learn quickly, generalize, and is stable. The CMAC ANN has been successfully applied to industrial applications, vibration control, robotic control, and fuel-injection systems [31-42].

Before the advantages of the CMAC are discussed in more detail, the disadvantages of the back-propagated multi-layer perceptron ANN for a real-time application will be discussed. First, the common back-propagated ANN is usually not feasible for on-line learning since numerous iterations are needed for the ANN to converge during training. Please note that on-line learning is not used for the HEUAV application but could be an area of future research. Second, many calculations are needed per training iteration for the back-propagated ANN which can necessitate custom hardware. Third, the more commonly used training algorithms for back-propagation are based on gradient techniques and the neural network can get "stuck" in a relative minimum during training vs. converging to the global minimum on the error surface as for the CMAC. Fourth, it has been shown that the local learning approaches used in the CMAC ANN and other associative memory networks are superior for control applications as compared to those used in the multi-layer ANNs [77]. Fifth, the computational time needed to produce an output from the CMAC ANN is minimal since only a few calculations are needed to obtain the output for an input. For these reasons, the CMAC was selected for the HEUAV application instead of the back-propagated multi-layer ANN.

3.2 History of the CMAC ANN

The CMAC neural network was originated by James Albus in 1975 [125, 126]. The CMAC is modeled after the method that the cerebellum uses to learn and store information and control reflexive movement. In contrast, a traditional neural network attempts to mimic the interactions between the brain's neurons. The CMAC attempts to duplicate the functional properties of the brain instead of the structure of it [126]. The CMAC was not called a neural network at the time it was proposed since the term was not commonly used then. Nevertheless, the CMAC can definitely be considered a neural network.

The CMAC is a lattice-based associative memory network (AMN) that nonlinearly maps the inputs to a hidden associative memory. The hidden memory is then linearly mapped to an adaptive weight vector that generates the output. The output is the sum of the activated weights. For each input, only a small subset of the network influences the instantaneous output which minimizes the computational time. This is a significant benefit for an embedded controller. Whereas the computational cost for most neural networks is exponentially dependent on the dimensions of the input space, the computational cost for a CMAC is linearly dependent on the input space dimensions [127]. In short, the CMAC takes real-valued vectors and produces real-valued output vectors, can learn locally and generalize, can learn nonlinear functions, has a relatively short training time, requires a small number of computations per training iteration, and can be implemented in simple software and hardware [29]. It is reported that the number of training iterations is orders of magnitude smaller than that of other ANNs [30, 128]. A possible disadvantage of the CMAC includes possible noise if hash coding is used to store the associative memory. For the HEUAV application, hash coding is not used due to the errors introduced into the control surface approximation. The CMAC ANN is applied to the control of the HEUAV propulsion system and will essentially store the efficiency and other maps for the engine, motor, and battery pack to produce commands for the propulsion system components.

The CMAC can be thought of as an adaptive look-up table. The CMAC is better suited to real-time control as compared to a look-up table for two reasons. The CMAC can generalize whereas a look-up table cannot. Also, for large input spaces, the CMAC requires much less memory than a look-up table. Both of these topics will be discussed in more detail later. The memory requirements for the CMAC ANN are analyzed for the HEUAV application in Chapter 6.

3.3 CMAC ANN Description

A diagram of a typical CMAC artificial neural network is shown in Figure 3-1. Continuous vectors are first transformed into quantized input vectors. The maximum and minimum values of the inputs (range) are needed along with the quantization width (precision or resolution) of the inputs to determine the size of the input space [32]. The input space is n-dimensional if more than one input is fed to the CMAC structure. Second, the input space is nonlinearly mapped into exactly L locations (generalization factor) in the associative memory satisfying the uniform projection principle. Please note that the nonlinear mapping in a CMAC structure occurs in the initial mapping and not in the sigmoid/threshold function of a neuron as in other types of ANNs. Each of the association cells (also referred to as basis functions with support or receptive fields) within each parallel layer of the memory has a corresponding weight. Each association cell has a support area of L^n where n is the dimension of the input space. Third, for each possible input, the weights corresponding to the L activated memory locations are then summed to form the output. The weights mapped to each memory location determine the output and are adaptively updated. The output is linearly dependent on the adaptive weights and therefore allows convergence conditions to be established [127]. It is emphasized that the nonlinear fixed mapping occurs from the input space to the association layers, and the adaptive linear mapping occurs from the association layers and the associated weights to the output.

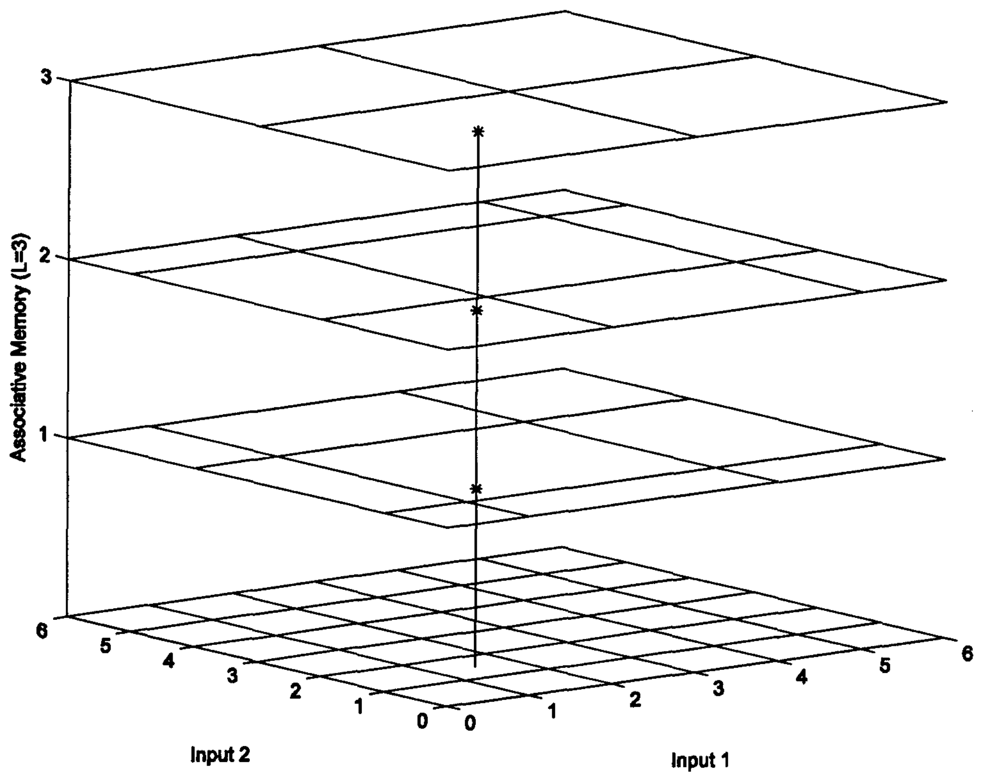


Figure 3-1: CMAC ANN Structure

The nomenclature used in this dissertation for the CMAC structure will be:

n: Dimensions of the input (dimensions of each lattice cell)

L:	Number of association layers (number of non-zero basis functions for each input)
b:	Total number of basis functions (association cells) in the association layers
N:	Total number of possible inputs (number of lattice cells)

These parameters typically have the following relationship for a controls application:

$$n \leq L < b < N \qquad \textbf{(3-1)}$$

The example in Figure 3-1 has parameters of n=2, L=3, b=22, and N=36. The total number of basis functions, b, is determined by summing the number of basis functions for each association layer. The total number of possible inputs, N, is found by multiplying the number of intervals, v_i, in each direction as follows:

$$N = \prod_{i=1}^{n} v_i \qquad \textbf{(3-2)}$$

where N is the number of entries that would be found in a look-up table. The CMAC is considered well defined if $1 \leq L \leq \max(v_i)$ for $1 \leq i \leq n$ [127]. If the CMAC is not well defined, then a basis function will cover a relatively large area of the input space (lattice).

The distribution of the layers is determined by a displacement vector. The original Albus scheme offset the layers by one from each other. The displacement vector is (1,1,1), (2,2,2), and (3,3,3) for the example in Figure 3-1. The original scheme satisfies the uniform projection principle but does not produce results as good as other schemes. Parks and Militzer produced tables of displacement vectors as a function of L and n [129]. Their proposed displacement vectors are based on the distance between the elements in the association vectors and elements in the input space. The placement of the overlays is optimized by directly relating the Hamming distance (absolute value of the differences of each component) between the association vectors to the Euclidian distance between the input values [30, 129]. The improved displacement vectors generally produce a smoother approximation for a function.

3.3.1 Training

Several techniques can be used to update the weights in the CMAC structure. For the results in this dissertation, the CMAC's weights are updated using an instantaneous gradient descent method to minimize the mean square error (MSE). The instantaneous estimate of the MSE is described using [32]:

$$E = \frac{1}{2} \cdot (y_{\text{Desired}} - y_{\text{CMAC}})^2 \tag{3-3}$$

where: E is the instantaneous estimate of the mean square output error

y_{Desired} is the desired output

y_{CMAC} is the output of the CMAC controller

Taking a partial derivative with respect to an activated weight, w_j:

$$\frac{\partial E}{\partial w_j} = -(y_{\text{Desired}} - y_{\text{CMAC}}) = -(y_{\text{Desired}} - \sum_{i=1}^{L} w_{i,\text{Activated}}) \tag{3-4}$$

Only local learning occurs since only the activated weights are updated. The resulting first-order update training rule using the instantaneous gradient descent method is:

$$w_{j,\text{updated}} = w_{j,\text{previous}} + \frac{\delta}{L} \cdot (y_{\text{Desired}} - y_{\text{CMAC}}) \tag{3-5}$$

where δ is the training rate [127]. The learning algorithm requires minimal memory and computational cost.

3.3.2 Convergence and Stability

The convergence of the network is critical during training. Miller states that the CMAC will converge to a global minimum on the error surface (if exact training rules are used) and since the error surface has no local minima, it can learn nonlinear functions at least approximately [130]. To better understand the convergence capabilities, the solution matrix needs to be described.

The solution matrix for the CMAC can be viewed as the solution to a system of linear equations:

$$\mathbf{y}_{\mathbf{Desired}} = \mathbf{A}\mathbf{w} \tag{3-6}$$

where: **A** is the matrix composed of the association vectors (Nxb)

w is the weight vector (bx1)

$\mathbf{y}_{\mathbf{Desired}}$ is the output vector or the control signal (Nx1)

The **A** matrix is sparse and does not have full column rank (columns are linearly dependent). Since the rank of **A** is equal to b or less and therefore does not have full rank for the CMAC ANN, **A** has a left inverse but it is not unique [131]. The ideal solution to Equation 3-6 can be obtained from the pseudo-inverse $\mathbf{A}^{+} = \mathbf{A}^{T}(\mathbf{A}\mathbf{A}^{T})^{-1}$. For the CMAC application, using the pseudo-inverse is not practical due to the size of the matrices. Since A is sparse and the convergence is rapid, an iterative method is used. Iterative techniques such as the instantaneous gradient descent method are used to train the CMAC ANN.

A theorem given by Brown and Harris states that the rank of the set of b binary basis functions of a well-defined CMAC is b-(L-1) [127]. In words, the rank of the set is equal to b minus the rank of the null space. The number of weights that can be assigned arbitrarily is L-1, but the L^{th} weight is not arbitrary so the rank of the null space is L-1. Although a global minimum exists on the error surface, more than one weight matrix can solve Equation 3-6. The resulting solution using instantaneous training rules can fall into a "minimal capture zone" which is an approximation to the global minimum [127, 132].

3.3.3 Generalization

The CMAC generalizes due to the width and overlap of the association cells in the hidden layers [126]. The generalization is determined by the initial nonlinear mapping, since each basis function or association cell has a pre-determined corresponding support or receptive field. The supports each have a volume of L^n or less if on the edge of the input space. Therefore, the generalization parameter, L, determines the number of association layers, the number of weights contributing to each output, and the size of support for each basis function [127]. If two inputs are relatively close to each other in the input space, then approximately the same association cells will be activated to produce an output. For two inputs that are spaced far apart, entirely different association cells are activated. The CMAC generalizes over a small area which minimizes the computations required for each training iteration. However, if L is large, the generalization is less local but the memory requirement is less.

3.3.4 Modeling Errors

Most neural networks are potentially universal function approximators if given sufficient resources. However, for practical applications, the function approximation will include modeling errors. For the CMAC neural network, the initial nonlinear mapping affects the modeling capabilities. The generalization parameter and the displacement vector also influence the modeling capabilities. As L increases, the generalization is less local and the modeling error typically increases. For a look-up table, L=1, and there is no error. The flexibility of the CMAC decreases as L increases due to the decreased generalization. The advantage is the decreased memory requirement. The instantaneous gradient descent training method also introduces errors into the modeling capability. The modeling errors introduced by the instantaneous training method can be reduced if the learning rate is allowed to decrease to zero as the training progresses.

Brown and Harris have investigated the types of functions the CMAC can and cannot model and have developed consistency equations and orthogonal equations, respectively [127]. The consistency equations provide insight into which functions the CMAC can model exactly, and the orthogonal equations show which functions the CMAC cannot model exactly. Any function can be decomposed into a set of consistency equations and a set of orthogonal functions. The decomposition will determine the modeling error for a particular function. For most applications, the modeling error is low even if the CMAC cannot model the function exactly if the appropriate parameters are chosen for the number of layers, the learning rate, and the structure of the hidden layers.

3.3.5 Memory Requirement

In addition to comparing performance and energy use between the rule-based and CMAC controllers, the microprocessor memory requirements are analyzed for the CMAC ANN controller in Chapter 6. A typical processor such as the Motorola MPC555 has approximately 512 KB of flash memory. If each CMAC weight is assumed to be a float value requiring 4 bytes (32 bits), the allowable size of the associative memory in the CMAC can be determined. The number of weights is approximately, $N/L^{(n-1)}$, where N is the possible number of inputs, L is the number of association layers, and n is the dimension of the input space [127]. To produce an output, the weights stored in L memory locations are summed. The CMAC structure greatly saves on memory and computational requirements as compared to having a multi-dimensional look-up table or computing a hyper-plane for every instant in time.

3.3.6 Higher-Order Basis Functions

The binary basis functions used in the original Albus CMAC description produce a piecewise constant output. Higher-dimensional basis functions can produce a smoother output, if required, but at the expense of computational time. Research into applying higher order basis functions to the CMAC ANN can be found in Lane et al. [133].

3.4 Function Approximation Example

To illustrate the use of the CMAC neural network to approximate a surface, a simple example is given. The generalization number will be increased to illustrate how the accuracy and memory requirement varies with a change in the number of association layers (generalization factor).

The function to be modeled is given by:

$$f(x,y) = e^{0.1\cdot(x^2+y^2)} \cdot \sin(2 \cdot x + y) \tag{3-7}$$

and the plot is shown in Figure 3-2.

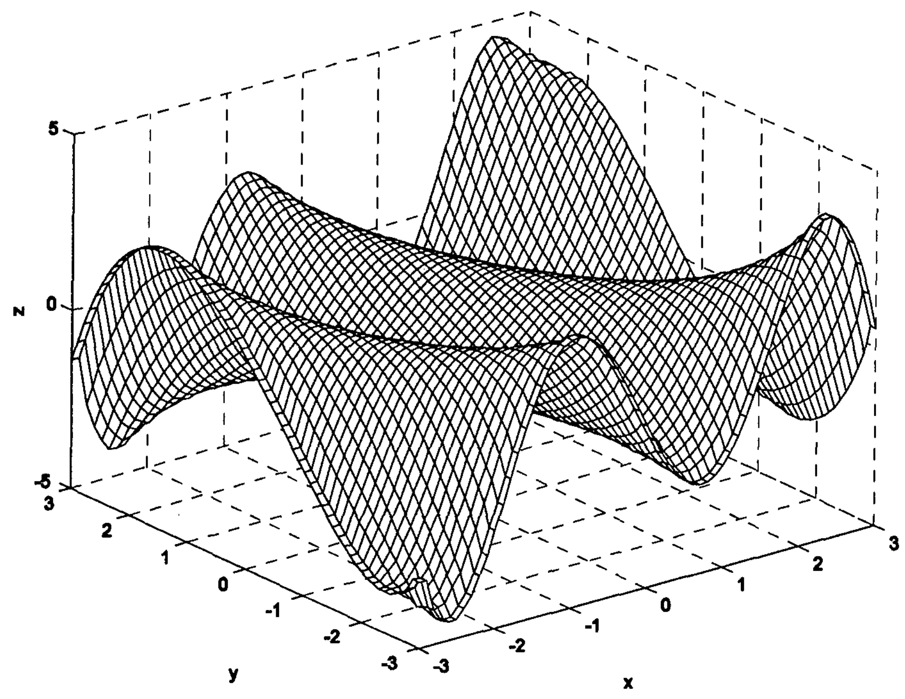


Figure 3-2: Original Function

In Table 3-1, the RMS error and the memory savings as compared to a look-up table (LUT) are listed. The training rate, δ, was equal to 0.05, and the number of training iterations was 250. As the generalization factor (number of layers in the neural network) increases, the RMS error generally increases as would be expected.

Table 3-1: RMS Error and Memory Savings for CMAC Approximations

Generalization Factor	RMS Error	Memory Savings[1]
3	5.71×10^{-2}	2.81
5	6.20×10^{-2}	4.40
8	7.59×10^{-2}	6.43
10	9.27×10^{-2}	7.60

[1]Number of entries in a LUT/number of weights in the CMAC associative memory

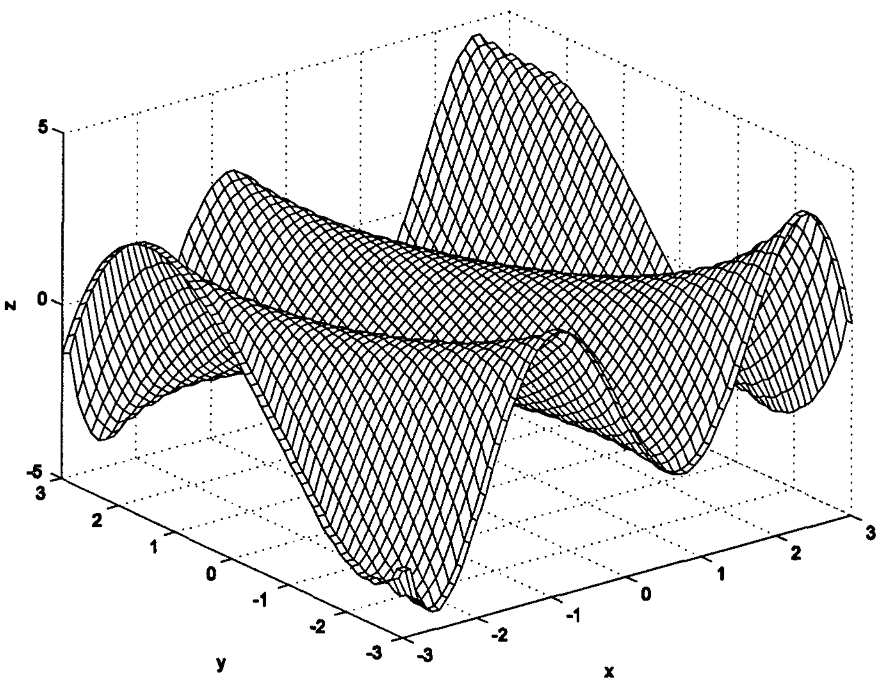


Figure 3-3: CMAC Approximation (L=3) of Original Function

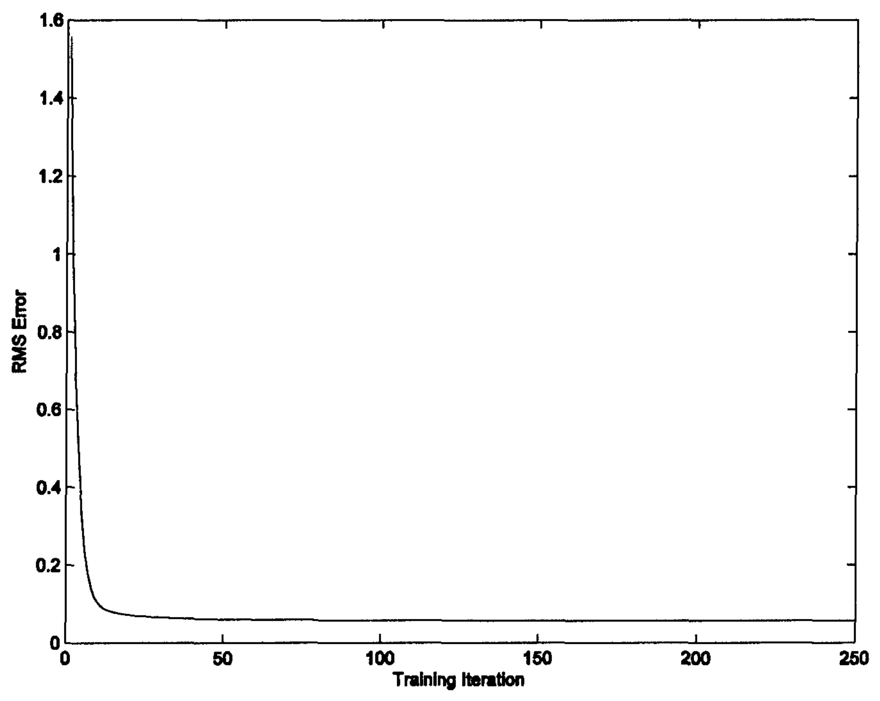


Figure 3-4: RMS Error for CMAC Approximation (L=3)

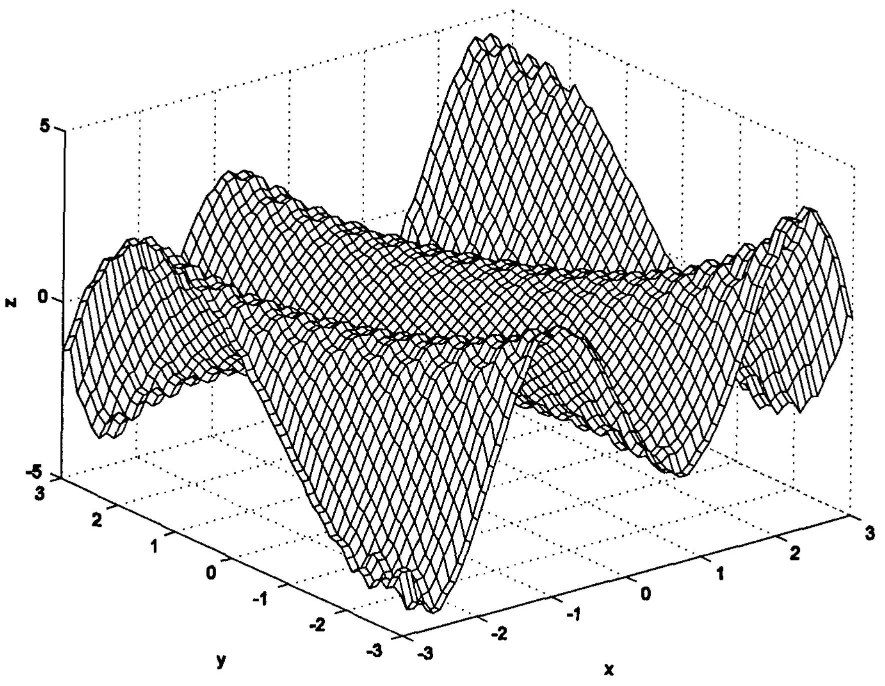


Figure 3-5: CMAC Approximation (L=10) of Original Function

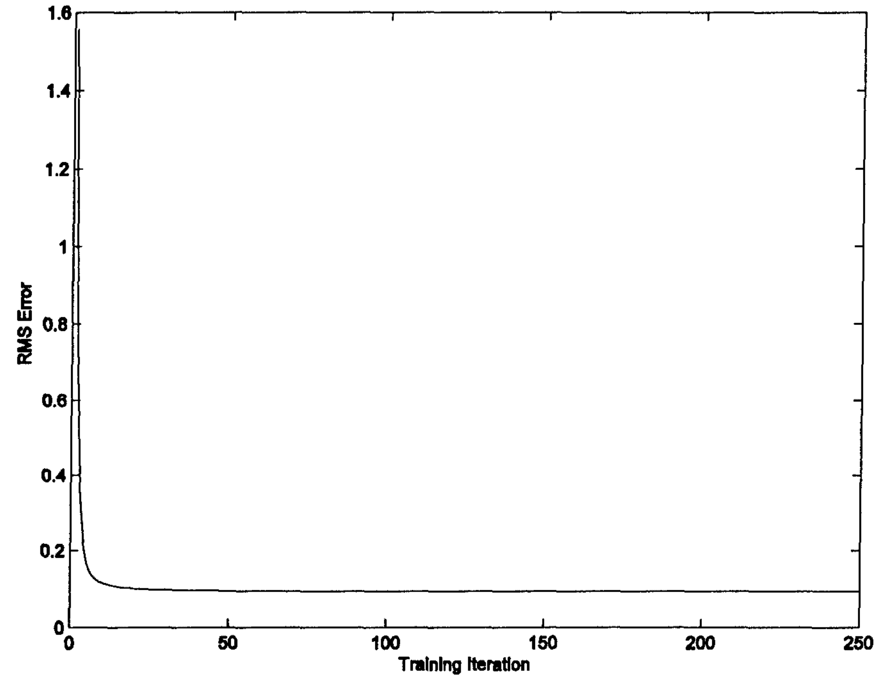


Figure 3-6: RMS Error for CMAC Approximation (L=10)

Figures 3-3 and 3-4 show the CMAC approximation to the function for a CMAC structure with L=3 and the corresponding RMS error during training, respectively. The number of hidden layers is one more than the number of inputs. The approximation is very close to the original function. Figures 3-5 and 3-6 show the CMAC approximation to the function for a CMAC structure with L=10 and the corresponding RMS error during training, respectively. As the number of association layers increases, the modeling error gets worse. The amount of acceptable error must be determined for an application which limits the amount of potential memory savings. A compromise must be made between the accuracy of the function approximation and the amount of memory savings.

BLANK

Chapter 4: Hybrid-Electric Unmanned Aerial Vehicle Simulink Model

A MATLAB/Simulink model of the parallel hybrid-electric unmanned aerial vehicle (HEUAV) was developed for the dissertation research. The model is derived from the Visteon/UCD hybrid-electric vehicle (HEV) model designed by the HEV Center under contract #ABG P0000 013588. The forward-facing simulation model was originally for the Coulomb parallel HEV. The model of the hybrid-electric propulsion system was modified to match the configuration and the dynamics of the HEUAV, but the overall hierarchy and structure of the model has been retained. Specifically, the subsystem model of the UAV replaces the model of the automobile, and the propeller model replaces the transmission model. However, the engine, clutch, electric motor, and rule-based controller models are very similar to the original subsystem models. A standard atmosphere model was added to account for changes in the density, temperature, and pressure with altitude. This chapter briefly explains the HEUAV model, and the improvements that were made to the original model during the dissertation research.

The structure of the simulation program is modular, and several of the Simulink blocks, or modules, in the original Visteon model were closely modeled after NREL's ADVISOR simulation program [134]. These blocks include the battery model and the speed profile model. These same blocks were used in the HEUAV model to minimize the software development and to use software components that have already been validated.

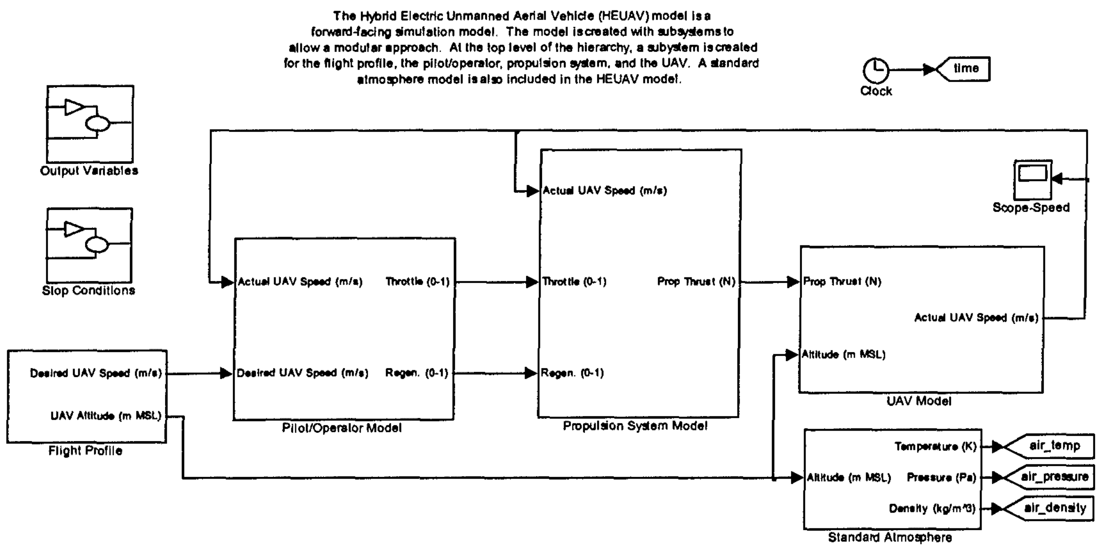

The Hybrid Electric Unmanned Aerial Vehicle (HEUAV) model is a forward-facing simulation model. The model is created with subsystems to allow a modular approach. At the top level of the hierarchy, a subystem is created for the flight profile, the pilot/operator, propulsion system, and the UAV. A standard atmosphere model is also included in the HEUAV model.

Figure 4-1: HEUAV Simulink Model

The HEUAV model uses empirical data, theoretical predictions, and dynamics principles to simulate the HEUAV. Empirical data is used in look-up tables to estimate the efficiency maps of components such as the electric motor, engine, and battery pack. The components are modeled using performance data for off-the-shelf components to simplify the design of an actual hardware model if the opportunity arises. Theoretical predictions and empirical data are used to determine the lift-drag curve of a typical UAV, the efficiency of the propeller, and the state-of-charge (SOC) of the battery pack. Dynamics principles are used to model the engine, clutch, electric motor, gear reduction unit, propeller, and the UAV.

The forward-facing simulation program for the HEUAV model is required to simulate and test control system strategies. The forward-facing program is more realistic of a dynamic system and can be used to develop real-time control strategies. Backward-facing simulation programs, such as NREL's ADVISOR, are useful for sizing components and for estimating fuel economy. Although a backward-facing simulation program is usually easier to use and the run times are shorter, the forward-facing program was used for all of the dissertation research to test the advanced control strategies.

The following sections document the blocks used in HEUAV Simulink model. Each block has corresponding data files that are loaded into the MATLAB workspace. The blocks are ordered as shown in Figure 4-1 starting at the flight profile, the most upstream component, then the pilot/operator, propulsion system, and ending at the UAV, the most downstream component. A standard atmosphere model is also included. The explanations in this appendix for the engine, motor, and clutch models are derived from the original model documentation [135].

4.1 Flight Profile

The flight profile model used in the HEUAV model reads in speed and altitude data as shown in Figure 4-2. The speed model is very similar to the speed profile or driving cycle used in ADVISOR [134]. The desired speed, in kts, of the UAV is read from the MATLAB workspace and is converted to m/s. A smoothing algorithm that uses centered-in-time averaging is available to smooth out jumps in the trace. For the HEUAV model, the speed is assumed to be the true velocity. In addition to the desired speed, the altitude of the UAV is read from the workspace and used in the simulation. Only the speed is used as a feedback variable since an increase in altitude with no throttle adjustments will result in a decreased speed. It is assumed that a longitudinal control system maintains the UAV at the correct altitude. The desired UAV speed and the altitude are outputs of the flight profile model.

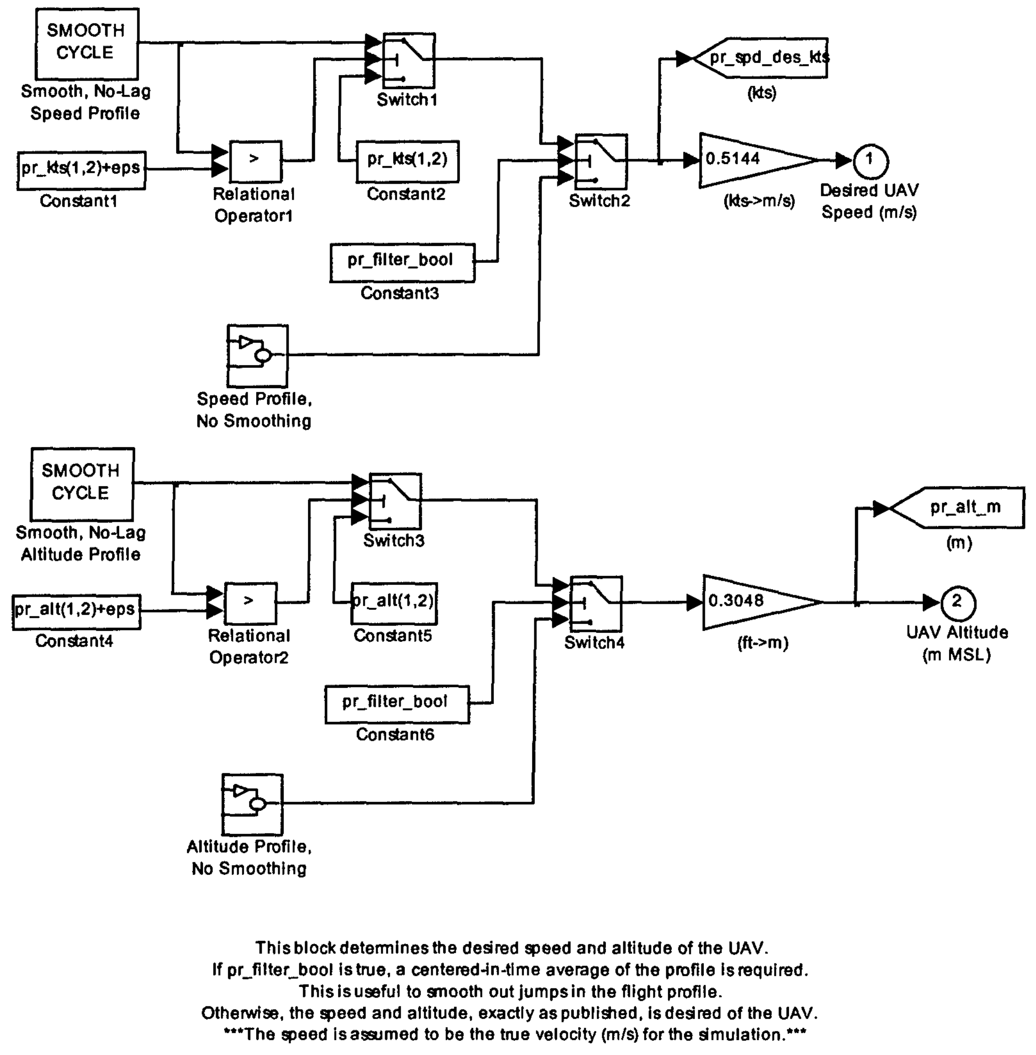

SMOOTH CYCLE
Smooth, No-Lag Speed Profile
pr_kts(1,2)+eps
Constant1
>
Relational Operator1
Switch1
pr_kts(1,2)
Constant2
pr_filter_bool
Constant3
Speed Profile, No Smoothing
Switch2
pr_spd_des_kts
(kts)
0.5144
(kts->m/s)
1
Desired UAV Speed (m/s)
SMOOTH CYCLE
Smooth, No-Lag Altitude Profile
pr_alt(1,2)+eps
Constant4
>
Relational Operator2
Switch3
pr_alt(1,2)
Constant5
pr_filter_bool
Constant6
Altitude Profile, No Smoothing
Switch4
0.3048
(ft->m)
pr_alt_m
(m)
2
UAV Altitude (m MSL)
This block determines the desired speed and altitude of the UAV.
If pr_filter_bool is true, a centered-in-time average of the profile is required.
This is useful to smooth out jumps in the flight profile.
Otherwise, the speed and altitude, exactly as published, is desired of the UAV.
The speed is assumed to be the true velocity (m/s) for the simulation.

Figure 4-2: Flight Profile Model

4.2 Pilot/Operator

The pilot/operator model shown in Figure 4-3 is almost identical to the UCD Visteon driver model except there is a regeneration command instead of a brake output as there would be for an automobile [135]. The model is relatively simple, but a significant amount of tuning was performed to achieve good performance over a wide operating range.

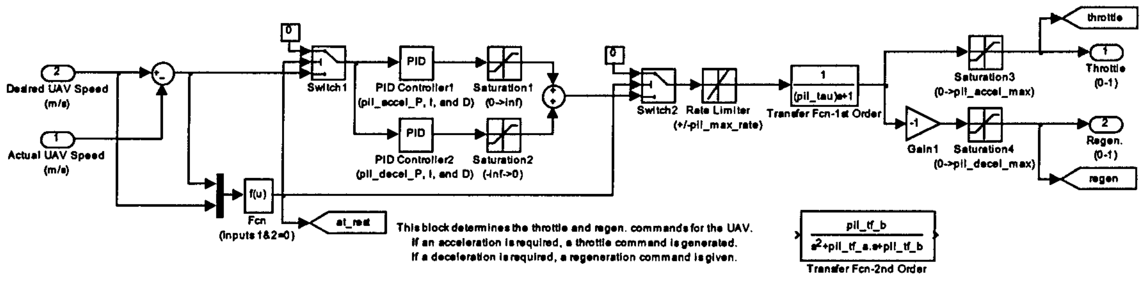

2
Desired UAV Speed (m/s)
1
Actual UAV Speed (m/s)
0
Switch1
PID
PID Controller1 (pil_accel_P, I, and D)
Saturation1 (0->inf)
PID
PID Controller2 (pil_decel_P, I, and D)
Saturation2 (-inf->0)
f(u)
Fcn (inputs 1&2=0)
at_rest
This block determines the throttle and regen. commands for the UAV.
If an acceleration is required, a throttle command is generated.
If a deceleration is required, a regeneration command is given.
0
Switch2
Rate Limiter (+/-pil_max_rate)
1 / (pil_tau)s+1
Transfer Fcn-1st Order
pil_tf_b / s²+pil_tf_a.s+pil_tf_b
Transfer Fcn-2nd Order
-1
Gain1
Saturation3 (0->pil_accel_max)
Saturation4 (0->pil_decel_max)
throttle
1
Throttle (0-1)
2
Regen. (0-1)
regen

Figure 4-3: Pilot/Operator Model

The inputs to the block are the desired speed and actual UAV speed. The desired speed is a direct input from the flight profile block. The actual speed is taken from the UAV model and is the true velocity of the UAV. The normalized outputs from the pilot/operator model are the throttle output (acceleration) and the regeneration command (deceleration).

The pilot/operator model has three different stages. The first stage is the implementation of proportional-integral-derivative (PID) controllers. The PID controllers use a modified version of the standard Simulink PID block. One PID controller is used for acceleration and another one for deceleration. Each one is saturated to only provide a contribution when it is required. The second stage adds the output of these controllers together and limits the rate of change of the combined response. A rate limiter prevents large signal changes from causing step outputs and a filter smoothes the signal. The third stage saturates the outputs that are passed to the rest of the HEUAV model. The throttle command (acceleration) is saturated so it is always between 0 and 1, and the regenerative command (deceleration) is multiplied by -1 and saturated so it is always positive between 0 and 1. A conditional statement ensures that the pilot/operator will output zero commands when the desired speed and delta are zero.

The block was hand tuned by previous researchers to meet the conventional traces used within ADVISOR. Specifically, the model was tuned so the actual vehicle speed would be within the standard specification of +/-2 mph of the desired speed trace such as FUDS, HWFET, and US06 [135]. The pilot/operator model was further tuned during the research, so the desired trace would be within several kts of the desired speed.

4.3 Propulsion System

The propulsion system block is shown in Figure 4-4. Within the block are the engine, clutch, battery, electric motor, gear reduction, propeller, and propulsion system controller blocks for the HEUAV. An engine block and controller for the original UAV configuration (engine-only) are also included.

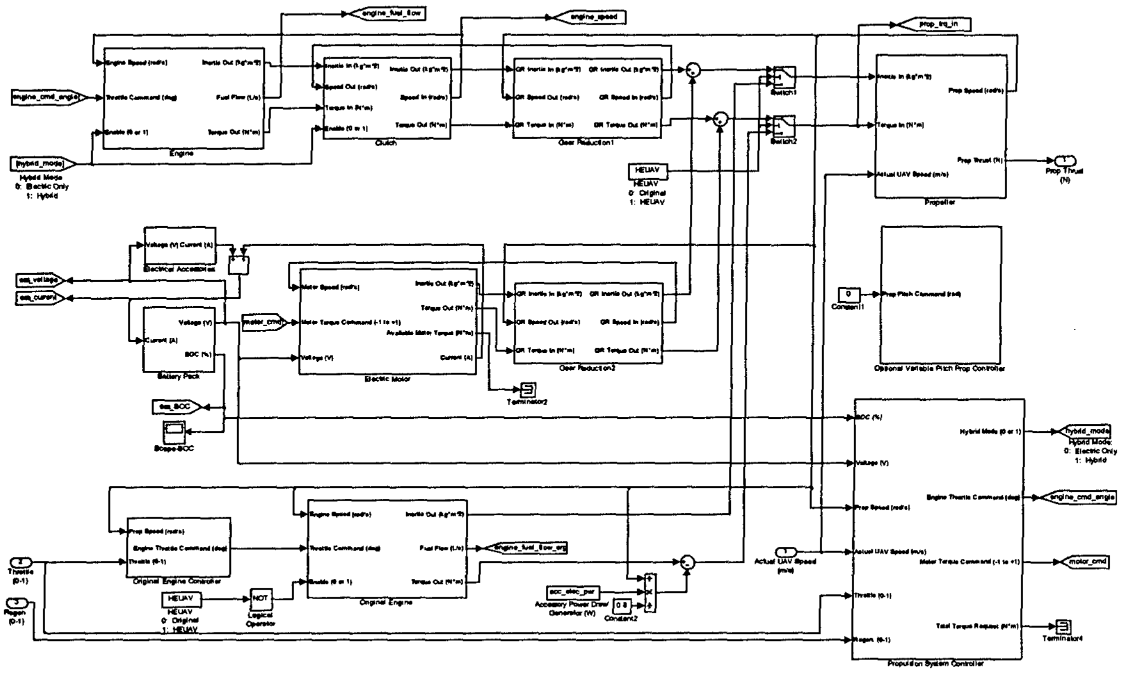


Figure 4-4: Propulsion System Model

4.3.1 Internal Combustion Engine

The engine is the furthest upstream component in the propulsion system. It constitutes the first value of inertia that is passed on to other components in the parallel hybrid-electric system. The engine requires a throttle position command and produces an output torque based on the speed determined by the propeller, gear reduction, and clutch blocks. The block also calculates the fuel flow and the fuel consumed.

The inputs to the engine block shown in Figure 4-5 include the enable (hybrid mode) command, speed in, and throttle position command. The outputs include the inertia, fuel flow, and torque out. Within the model, look-up tables for the throttle/torque relationship and fuel consumption are used to determine the outputs.

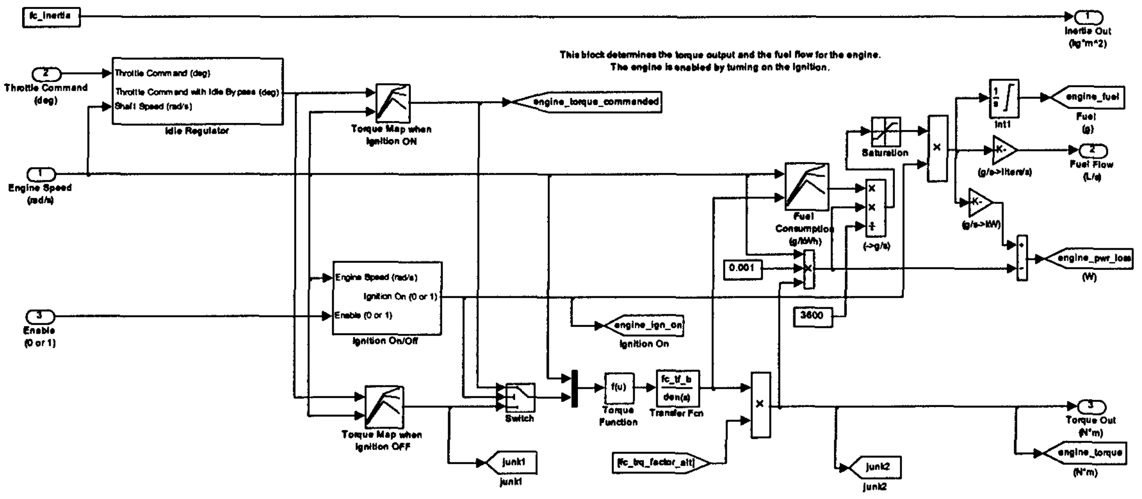


Figure 4-5: Engine Model

Several conditions will cause the engine model to disable the ignition. If the engine speed exceeds its maximum allowable speed or falls below a designated stall speed, the ignition is turned off. These limits prevent the engine from operating in these extreme regions. On most gasoline engines for model airplanes, there is no ignition shut off at high speeds since the engine is supposed to be matched to the propeller size and expected propeller loading. The maximum operating limit is included in the HEUAV model as a check to be sure the engine does not operate too high if the clutch is disengaged at an inappropriate time, especially during propulsion control system design. The ignition shut off could easily be programmed into a real-time propulsion system controller. The stall speed limit is realistic since model airplane engines are programmed to turn off the ignition and shut off the fuel simultaneously when the engine is turned off.

The engine fuel consumption maps in NREL's ADVISOR return the fuel flow as a function of torque and speed [134]. In order to more accurately simulate a real engine, a throttle model was implemented by previous researchers. This allows the model to react to the throttle position and permits a more accurate representation of certain phenomena such as torque absorption at low throttle openings and high rotational speeds. Since torque data as a function of throttle and speed are not currently available, two formulas were created by previous researchers that estimate these maps [135]. The Visteon model documentation has more details on these maps [135]. The hypothetical throttle maps could be replaced with actual engine data if it became available.

The efficiency for the engine is inherent in the fuel consumption map. For a given torque and speed, a fuel consumption value is provided. The fuel consumption, in g/kWh, is used to calculate the fuel used and the fuel flow. The output torque is adjusted using a first-order approximation based on the decrease in density with altitude. The power loss is equal to the power produced from the fuel minus the current power output of the engine.

The transient effects on the efficiency and fuel consumption maps are not considered in the model. The lower density effect with increasing altitude is considered by using a first-order approximation for the decrease in density, ρ/ρ_{SL} [136]. The small HEUAV is not expected to fly above several thousand feet MSL which should not effect the fuel consumption considerably so the first-order approximation is sufficient.

4.3.2 Clutch

The clutch model is downstream from the engine in the propulsion system block (see Figure 4-6). The clutch model for the Visteon simulation was designed by previous researchers using basic, well-known behavior models [135]. The equations are parameterized with data that are relatively easy to obtain or estimate.

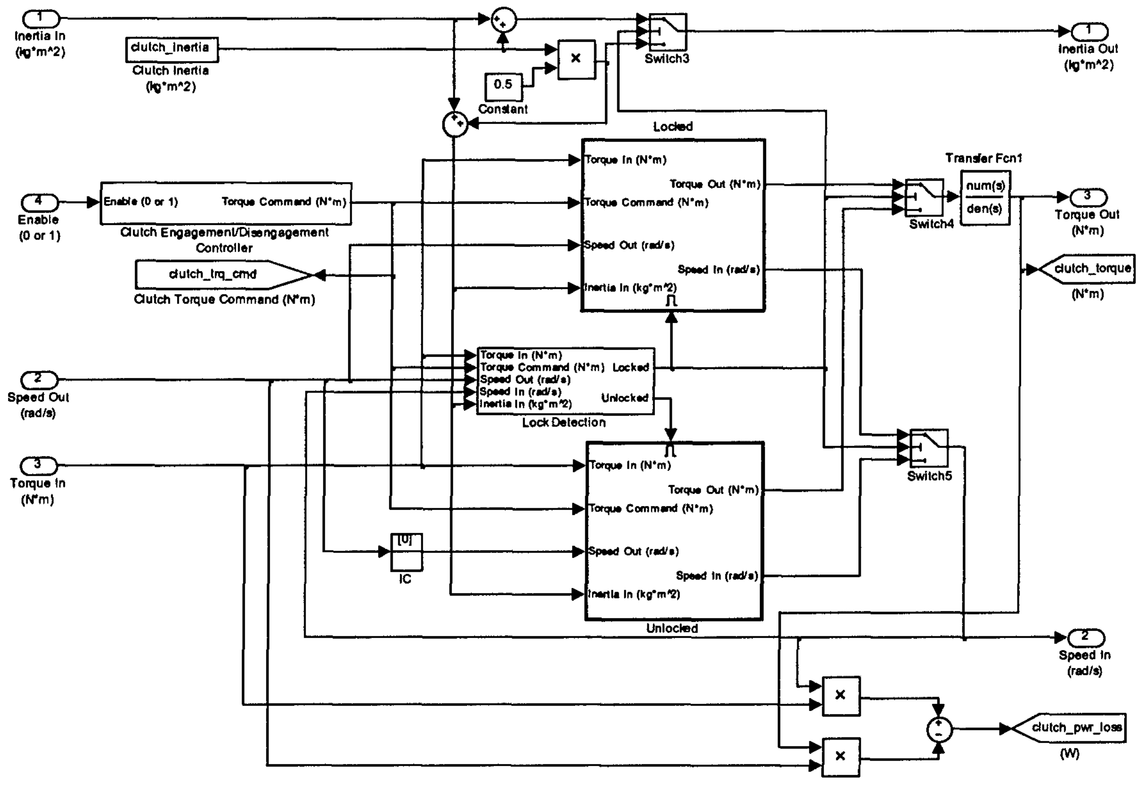


Figure 4-6: Clutch Model

The inputs to the clutch model are enable (hybrid mode), inertia in, speed out, and torque in. The enable input is the main command for this block and determines whether the clutch is being engaged or disengaged. If the enable command is greater than 0.5, the clutch is commanded to engage. If it is less than 0.5, the clutch is commanded to disengage. Each command is followed by ramping the torque demand of the clutch, so the transition will take a certain amount of time as specified in the input file. The inertia input is the upstream inertia from the engine. Depending on the internal conditions, this inertia will be passed downstream or used to determine the internal acceleration. The speed out is determined by the downstream components and determines the speed to be passed upstream. The torque in is obtained from the upstream components. When the clutch is locked, this value is passed directly to the torque out. When the clutch is open or slipping, this torque is used with the inertia to determine the acceleration of the upstream components.

The outputs of the clutch model block are inertia out, speed in, and torque out. The inertia out is determined from the clutch state and the input inertia. If the clutch is locked, the input inertia is passed directly. If the clutch is disengaged or slipping, the output inertia is estimated to be one-half of the clutch inertia.

The speed in depends on the internal state. If the clutch is locked, the speed in is equal to the speed out. If the clutch is open or slipping, the speed in is determined by dividing the torque (input torque commanded or transfer torque) by the input inertia and integrating. During a transition from the locked state to the unlocked state, the state of the integrator is initialized with the current output speed. This behavior allows the engine to accelerate and decelerate in an unlocked state just as a normal engine would.

The torque out is determined by the input torque and the internal transient torque transfer magnitude. If the clutch is locked, the torque is transferred through the device. If the clutch is open, no torque is transmitted. If the clutch is slipping, the torque transferred through the device will be controlled and will ramp up over time until lock is achieved.

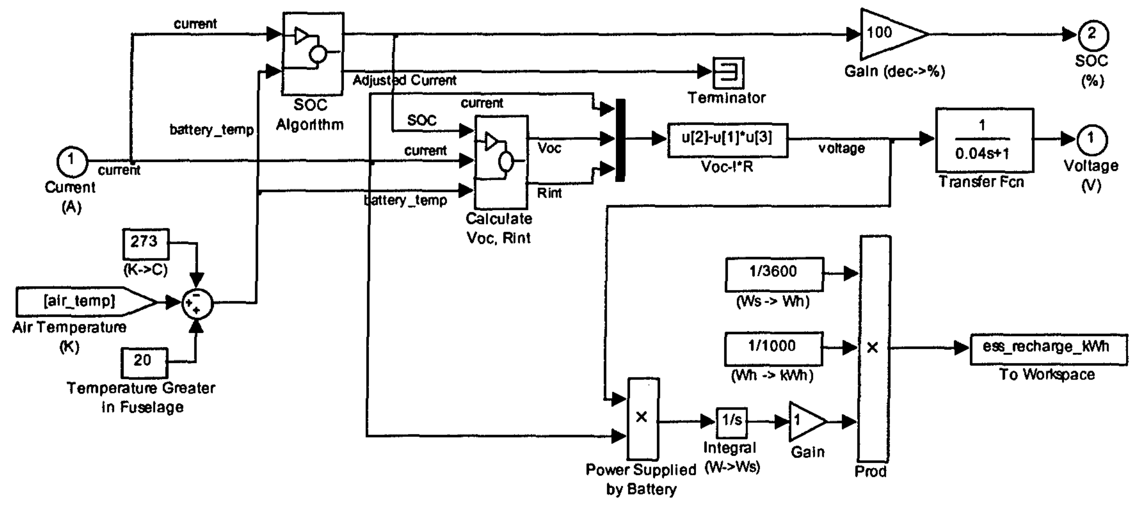
The clutch model can be roughly divided into four modules implementing two states. The modules are the clutch engagement/disengagement controller, the lock detector, the locked state processor, and the unlocked state processor. The Visteon model documentation has more details on these modules [135].

For the HEUAV, an electromagnetic clutch was modeled. The power drain of an electromagnetic clutch is not modeled but could easily be lumped in with the accessory loads on the batteries. Empirical data for a potential electromagnetic clutch uses 5-6 W.

4.3.3 Battery Pack and Electrical Accessories

The battery pack model shown in Figure 4-7 determines the electrical energy use of the motor and electrical accessories. The block models the energy storage system as a voltage source in series with an internal resistance and determines the effect of current draw on the voltage and the SOC. The model was adapted from ADVISOR with only minor changes such as the change in ambient temperature with an increase in altitude and the amount of recharging needed to recharge the battery pack to its original SOC [134].

The battery pack block models the energy storage system as a voltage source in series with an internal resistance. The voltage source is dependent on the SOC and the ambient temperature. The internal resistance is dependent on the SOC, ambient temperature, and the current direction. The convention used is that the positive current represents the discharge current. The power loss during discharge is computed as $I^2 \cdot R$ and the recharge loss is due to the Coulombic inefficiency. Thermal modeling of the battery performance is modeled in the open-circuit voltage, internal resistance, and capacity parameters. The SOC is computed assuming that the Ah battery capacity is dependent on the ambient temperature. A first-order transfer function is included in the voltage line to smooth out the battery pack dynamics.



This block models the energy storage system as a voltage source in series with an internal resistance.
The voltage source is dependent on the SOC and the ambient temperature.
The internal resistance is dependent on the SOC, temperature, and current direction.
The SOC is computed assuming the Ah capacity is dependent on the ambient temperature.

The convention used is that the positive current represents the discharge current.
The power loss during discharge is computed as I^2*R and
the recharge loss is due to the Coulombic Inefficiency.

Thermal modeling of the battery performance is modeled in the
Voc, Rint, and capacity parameters.

Figure 4-7: Battery Pack Model

The electrical accessories component model includes all of the electrically-powered systems and the power demanded by the payload. The payload could include items such as infrared or video sensors or signal processing cards. The current drain for the accessory loads is determined by using a constant electrical accessory power load, dividing by a constant efficiency, and then dividing by the pack voltage.

4.3.4 Electric Motor

The motor model for the UCD/Visteon simulation was designed by previous researchers using parametric data in a series of simple equations [135]. The motor block uses manufacturer data and can be modified as necessary to tune the behavior of the model.

The inputs to the motor block are the torque command, motor speed, and voltage. The normalized torque command is the external request from the propulsion system controller. The motor controller is assumed not to have a direct torque input but a normalized input that is proportional to the maximum available torque. At low speeds in the torque-limited region, the input is a torque command. At higher speeds in the power-limited region, the input signal is scaled to make full scale correspond to the power-limited maximum torque. The input voltage and the motor speed are the physical inputs from the other modeled components. The input voltage to the motor controller is determined by the characteristics of the battery pack. The voltage for a given load is related to the current drawn by the motor controller to achieve a desired motor output power. The other input, motor speed, is determined by the gear reduction block. The propulsion system uses the torque of the engine, motor, and propeller, as well as the propulsion system inertia to determine the acceleration that is then integrated to determine the motor and engine speed. The motor speed is used by the motor block to determine the location on the efficiency map.

The outputs from the motor block are the motor inertia, available motor torque, torque out, and current. The inertia and torque out are used by the downstream components to determine the acceleration that is integrated and fed back as the motor speed taking into consideration the gear reduction unit. The current output is used to determine the battery SOC.

The electric motor module, as shown in Figure 4-8, was built by previous researchers to be modular and flexible while incorporating the safety features expected for a motor [135]. This model is intended to simulate an actual motor/controller system while retaining the ability to simulate different types of motors with incomplete data. The model was based on parametric data obtained from manufacturers for motor/controller systems.

The parametric model has three main sections. The first section is the safety limiter that implements the required safety limitations such as the maximum speed, current, and voltage. The second section is a filter on the limited torque request to approximate the torque delivery dynamics. The third section determines efficiency data based on the torque required, speed, and voltage. The Visteon model documentation has more details on these sections [135].

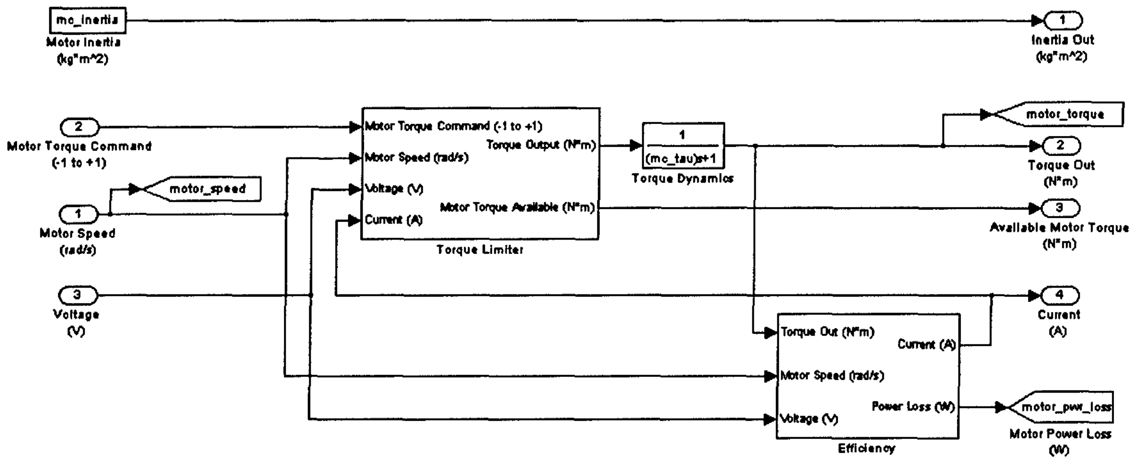


Figure 4-8: Electric Motor Model

The motor model uses a simplified dynamics model that implements important dynamics while preventing unnecessary complexity. Typical motor models use an armature inductance and inertia to create a second-order response. The dynamics model implemented is similarly divided into a mechanical piece and an electrical piece. The mechanical dynamics of the motor are modeled as inertia. This inertia is passed to downstream components. The electrical system dynamics are lumped into a pre-filter on the torque response. This allows the limiter block and the efficiency calculator to receive signals that do not change instantaneously and allows the electrical system dynamics to be changed rapidly with predictable results. The electrical dynamics are very dependent on the motor/controller system so it is difficult to create a universal filter. These dynamics are currently implemented as a first order filter as shown below:

$$\frac{T_{out}}{T_{in}} = \frac{1}{\tau \cdot s + 1} \tag{4-1}$$

where τ is the time constant. This provides a delay between the input and output while allowing the limiters to operate at full speed and avoid possible instabilities.

The efficiency calculation performed is a parametric model that determines the required input power from the speed, torque, and voltage of the motor. The data generally available for efficiency calculations are in the form of a map of efficiency vs. speed and torque. Multiplying the speed and torque, the output power can be calculated and used with the efficiency to determine the required input power. The efficiency map is also used to determine the regeneration torque by simply flipping the map about the zero-torque axis to form positive and negative torque regions.

4.3.5 Propulsion System Controller

The propulsion system controller is a supervisory system that issues commands to the engine and electric motor based on the pilot/operator request and other parameters. A block for an optional variable-pitch propeller is included in the model, and the controller could also produce commands for this optional propeller, if desired. The HEUAV model includes an option to switch between a rule-based strategy that is similar to those used in the UCD hybrid-electric vehicles and the neural network controller (see Chapter 6).

The inputs to the propulsion system controller include the battery SOC, actual UAV speed, propeller speed, throttle, and regeneration commands. The battery SOC input is essential for a hybrid-electric vehicle, especially when operating in a charge-sustaining mode. The rule-based controller in the model implements a charge-depleting mode and a charge-sustaining mode depending on the desired mission.

The propeller speed is used as the input to the controller since it is directly related to the motor speed. The engine speed is either zero or directly related to the propeller speed at all times except when the clutch is engaging or disengaging. The speed signal is used as an input to look-up tables that retrieve engine and motor data. The throttle and regeneration inputs are signals that range from zero to one and are sent by the pilot/operator model.

The outputs include the hybrid mode (engine enable and clutch engagement command), engine throttle command, and motor torque command. The engine enable command is a binary signal that commands the engine ignition to turn on or off. The same signal is also used to engage or disengage the clutch. The engine throttle command is a throttle position command to the engine. The electric motor command varies between -1 to +1 where negative values represent regeneration and positive values represent positive torque. The signal represents a fraction of the maximum torque of the motor at its current speed. More specifically, the normalized output represents the desired motor torque divided by the maximum motor torque provided by the speed-dependent look-up table.

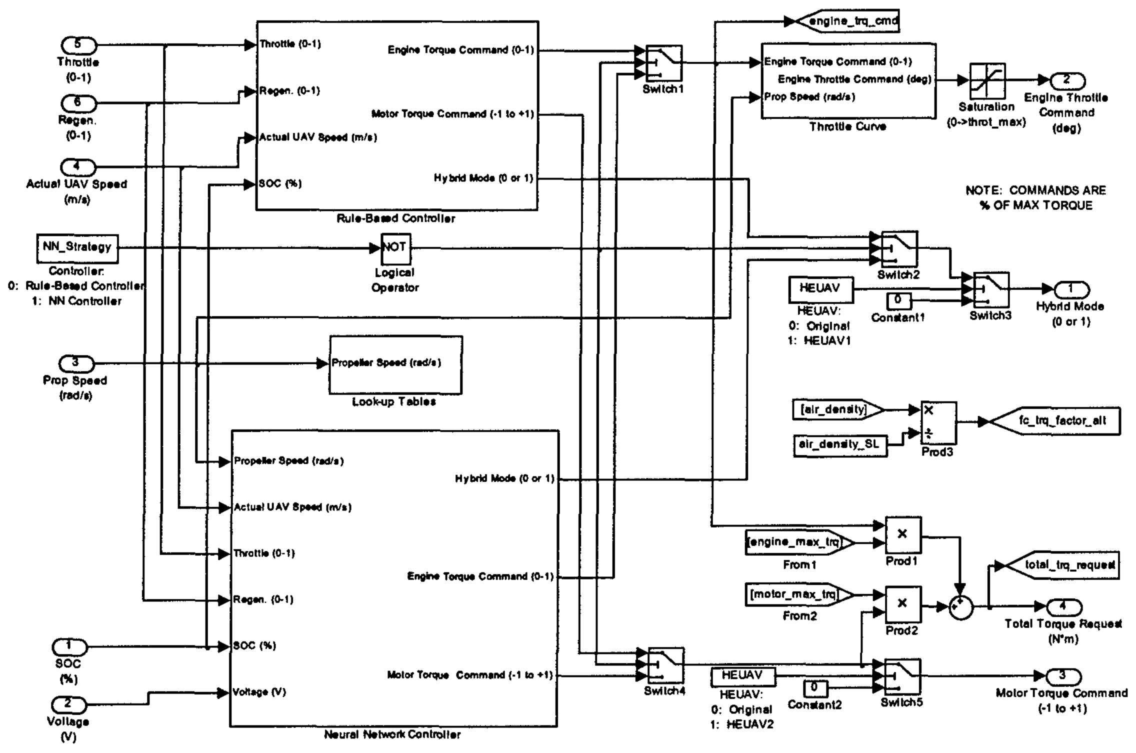


Figure 4-9: Propulsion System Controller Model

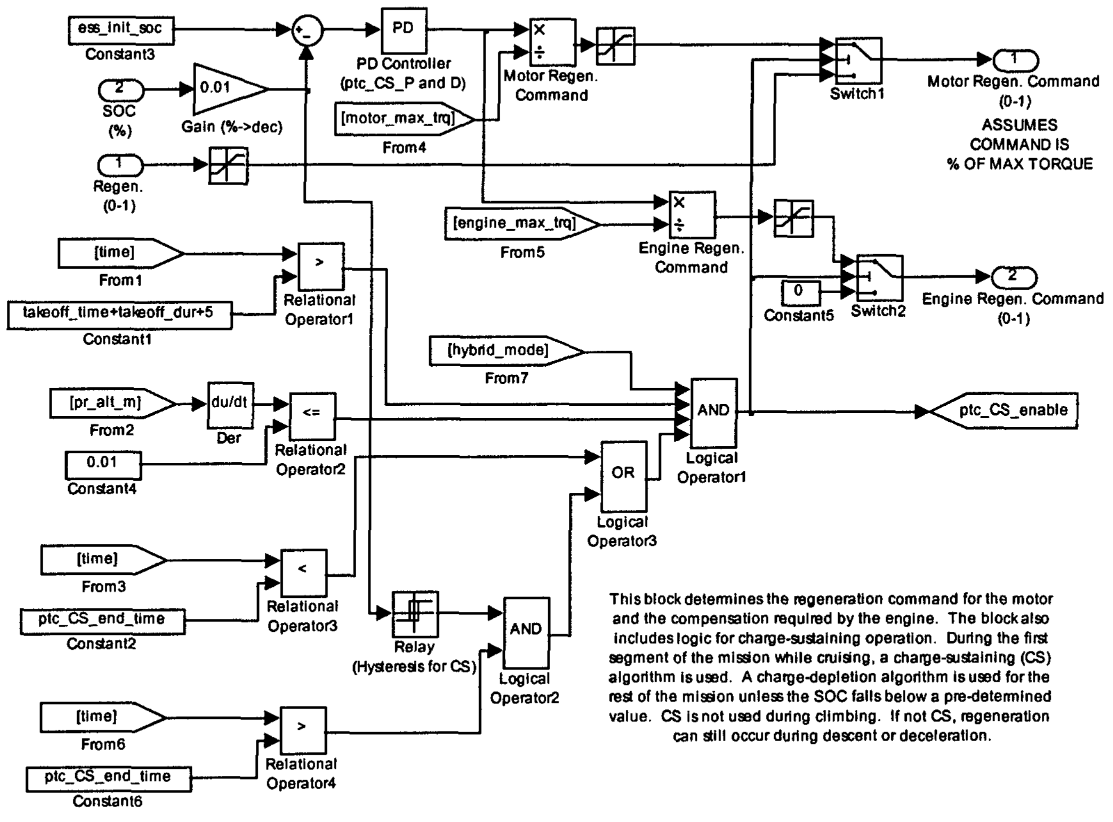
The propulsion system controller model is separated into several blocks with specific functions as shown in Figure 4-9. All control parameters, including look-up tables, are placed in the MATLAB workspace via data files (m-files). The original rule-based propulsion system controller model utilizes three major subsections: the throttle interpretation (resulting in torque commands), look-up tables based on the propeller speed, and a block that converts the engine torque command into a throttle position command. An additional block is provided for the neural network controller (see Chapter 6).

The look-up tables are necessary to properly calculate the engine and motor torque commands based on the operator's input. The look-up tables include the maximum engine torque, ideal operating line (IOL) torque for the engine, maximum motor torque, and the maximum regeneration torque. The tables are one-dimensional and relate the appropriate speed in rad/s to their respective output.

The rule-based controller block determines the output torque commands for the engine and the electric motor including regeneration. The engine command is between 0 and 1, and the motor command is between -1 and 1 to include regeneration. The fractions represent the fraction of each component's maximum torque. The engagement state of the engine and clutch, or "hybrid" mode, is based on operator or mission-dependent commands and the UAV speed. The rule-based controller assumes that the throttle position linearly commands a fraction of the total available propulsion system torque whether or not the engine is currently engaged. This allows the throttle response to be consistent. The throttle operation is based on two regimes: electric-only (engine-off) mode and hybrid-electric vehicle (engine-on) mode. In electric-only mode, the throttle position commands the electric motor torque. In the HEV mode, engine torque is commanded with increasing throttle position up to the engine IOL, then motor torque up to the maximum motor torque, and finally commands the remaining engine torque.

Regeneration using the electric motor occurs in a similar manner during deceleration or a descent. The regeneration input commands a fraction of the total regeneration torque available from the electric motor. Charge-sustaining operation for the controller is implemented in the same block as the regeneration command as shown in Figure 4-10. Charge-sustaining can be implemented for the first phase of the mission, and then charge-depletion can be used until the battery SOC drops to a pre-determined level.

The throttle curve block converts the desired engine torque to the respective throttle position. This calculation is performed using the mathematical relationship established in the engine data file.



This block determines the regeneration command for the motor and the compensation required by the engine. The block also includes logic for charge-sustaining operation. During the first segment of the mission while cruising, a charge-sustaining (CS) algorithm is used. A charge-depletion algorithm is used for the rest of the mission unless the SOC falls below a pre-determined value. CS is not used during climbing. If not CS, regeneration can still occur during descent or deceleration.

Figure 4-10: Regeneration/Charge-Sustaining Logic Block

4.3.6 Gear Reduction Unit

The gear reduction units connect components of the propulsion system to the propeller. This block calculates the transfer of engine or motor inertia, torque, and speed through the unit and adds torque loss to the system.

The inputs to the gear reduction unit are inertia in, torque in and speed out. The outputs are inertia out, torque out, and speed in.

Several simplifying assumptions were made in the design of the gear reduction block. The effect of the gear ratio (ratio squared) on the upstream inertia is included in the model, but the inertia of the gears is not included. The model does not incorporate any dynamic effects but only reduces the speed and increases the torque of the propulsion system.

4.3.7 Propeller

The propeller serves to provide thrust to propel the UAV through the air. The propeller block uses the standard advance ratio to determine the efficiency of the propeller and the thrust and power coefficients [137]. The propeller thrust and speed can then be determined. The advance ratio is calculated using:

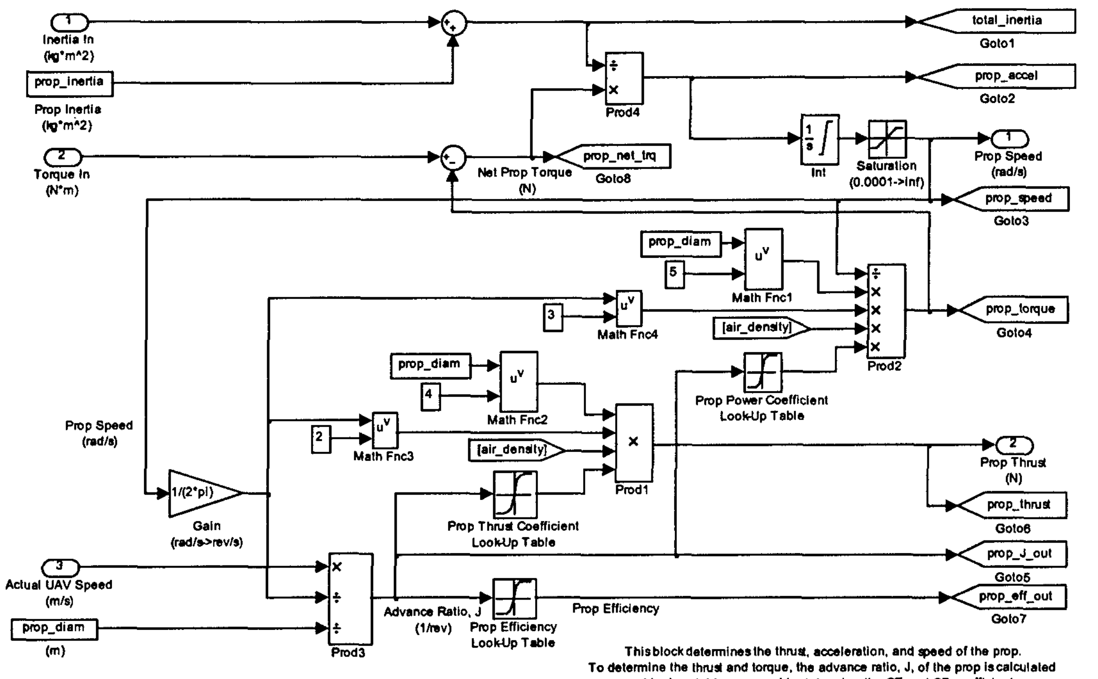
$$J = \left(\frac{V}{n \cdot D} \right) \tag{4-2}$$

where V is the true velocity (m/s) of the aircraft, n is the rotational speed (rev/s) of the propeller, and D is the diameter (m) of the propeller. The peak efficiency of the propeller is approximately 75% for cruise speed and is matched to the expected load on the propeller and the size of the engine. The thrust and the torque of the propeller are dependent on the advance ratio and are calculated using the following equations [138]:

$$\text{Thrust} = \rho \cdot n^2 \cdot D^4 \cdot C_T$$
$$\text{Torque} = \frac{\rho \cdot n^3 \cdot D^5 \cdot C_P}{2 \cdot \pi \cdot n} \qquad \textbf{(4-3)}$$

where C_T is the thrust coefficient and C_P is the power coefficient. Empirical data is used for the coefficients in the simulations.

The propeller block is shown in Figure 4-11. The inputs are the inertia in, torque in, and the actual UAV speed. The outputs are the propeller speed and thrust.



This block determines the thrust, acceleration, and speed of the prop.
To determine the thrust and torque, the advance ratio, J, of the prop is calculated and look-up tables are used to determine the CT and CP coefficients.
The acceleration of the prop is determined by calculating the net prop torque and then dividing by the power train inertia. The acceleration is integrated to get the prop speed.
A saturation block is used to prevent a negative prop speed.

Figure 4-11: Propeller Model

The propulsion system inertia from the upstream components is added to the propeller inertia to calculate the total inertia. The propeller inertia is calculated assuming the propeller can be modeled as a thin slender rod using:

$$I = \frac{1}{12} \cdot m \cdot D^2 \tag{4-4}$$

where m is the mass (kg) and D is the diameter (m) of the propeller. The total inertia of the propulsion system and the net torque of the propeller are used to calculate the acceleration of the propeller and the propulsion system. The acceleration of the propeller is integrated to obtain the propulsion system rotational speed.

4.4 Unmanned Aerial Vehicle

The unmanned aerial vehicle (UAV) model shown in Figure 4-12 implements basic equations typically used to model aircraft drag and lift. The model is intended to be modified for the intended UAV, so it is parameterized with commonly available data.

The inputs to the UAV model block are the propeller thrust and the altitude. The propeller thrust is calculated in the propeller block, and the altitude is obtained from the flight profile. The output of the UAV block is the actual UAV speed. The UAV speed (m/s) is saturated to prevent a negative speed. The forces acting on the UAV and the net propeller thrust are also outputs made available in the MATLAB workspace.

The equations of motion for the UAV in translational flight are needed to determine the acceleration and speed of the UAV [137]. For the simulations, the UAV is treated as a particle with mass and the two equations that are used include four forces acting on the UAV:

$$\begin{gathered} L - W \cdot \cos(\text{climb angle}) = 0 \\ T - D - W \cdot \sin(\text{climb angle}) = m \cdot \frac{dV}{dt} \end{gathered} \tag{4-5}$$

where: L=lift (N)

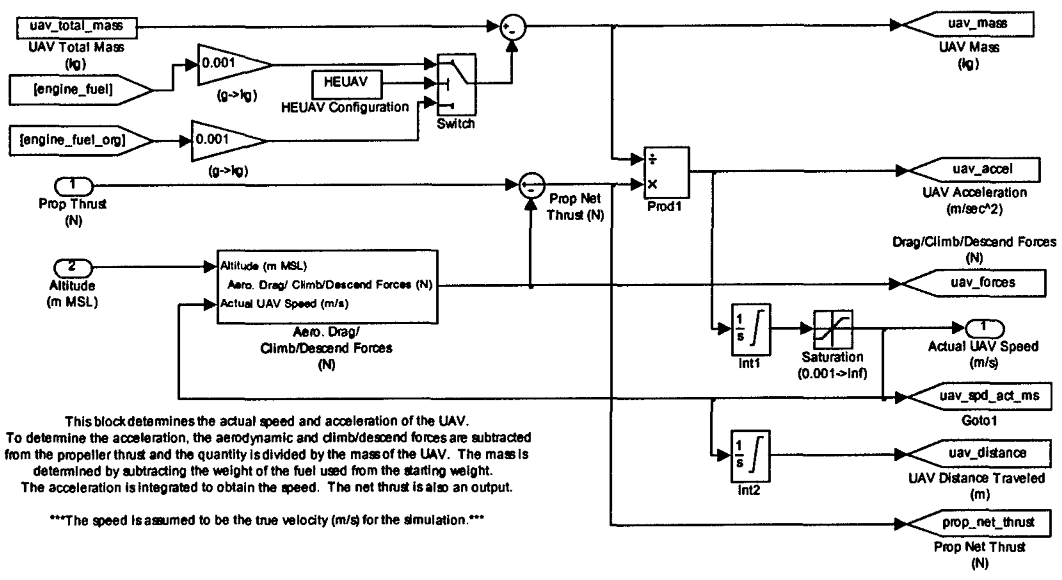
W=weight (N)

T=propeller thrust (N)

D=total drag (N)

m=mass of UAV (kg)

dV/dt=UAV acceleration (m/s^2)



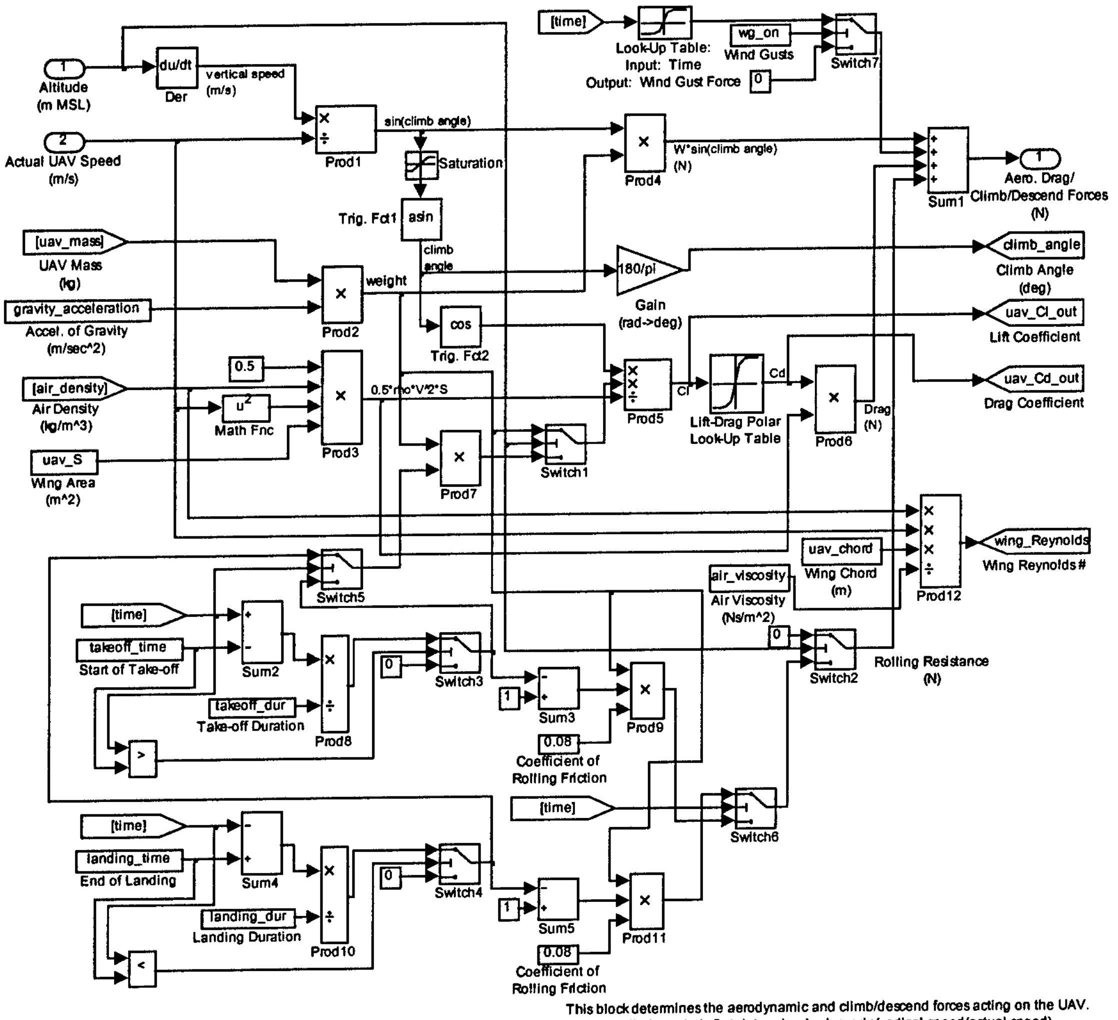
This block determines the actual speed and acceleration of the UAV.
To determine the acceleration, the aerodynamic and climb/descend forces are subtracted from the propeller thrust and the quantity is divided by the mass of the UAV. The mass is determined by subtracting the weight of the fuel used from the starting weight.
The acceleration is integrated to obtain the speed. The net thrust is also an output.

The speed is assumed to be the true velocity (m/s) for the simulation.

Figure 4-12: UAV Model

The first equation is the summation of the forces acting on the UAV perpendicular to the flight path. The second equation is the summation of forces acting parallel to the flight path. The assumptions made in the derivation of the two equations include: the thrust is along the fuselage reference line and the angle-of-attack is small. These assumptions are sufficient for performance analysis [137]. The second equation is implemented in the UAV model block to determine the acceleration of the UAV. The first equation is used to determine the lift coefficient and is implemented in the UAV forces block.

The vehicle model contains a block to determine the aerodynamic, climbing/descending, and rolling resistance (during take-off and landing) forces acting on the UAV. The inputs to the UAV forces block shown in Figure 4-13 are the UAV speed and the altitude. The output is the total forces acting on the UAV.



This block determines the aerodynamic and climb/descend forces acting on the UAV.
The climb angle is first determined using asin(vertical speed/actual speed).
The climb/descend force is calculated using W*sin(climb angle).
The aerodynamic force is calculated using L=W*cos(climb angle) and then using L
in the following expression for the lift coefficient: Cl=L/(0.5*rho*V^2*S). A table look-up
is then used to find the drag coefficient, Cd, from the lift-drag polar plot.
Rolling resistance and partial lift during takeoff and landing are also included.

Figure 4-13: UAV Forces Model

The forces block first determines the climb angle of the UAV using asin(V_v/V) where V is the velocity of the UAV and V_v is the vertical velocity of the UAV determined by taking the derivative of the altitude. The climbing/descending force is then calculated using W·sin(climb angle). The aerodynamic drag force is more complicated but uses well known equations. A drag polar in the form of a look-up table is used to determine the drag coefficient indexed by the lift coefficient. To better understand the drag polar, a brief overview will be given on the aerodynamic drag acting on the UAV. The total aerodynamic drag acting on the UAV includes two terms: the parasite drag (friction and pressure drag) and the induced drag (due to lift) [137]. The total drag can be expressed as:

$$C_D = C_{D,o} + \left(\frac{C_L^2}{\pi \cdot e \cdot AR} \right) \tag{4-6}$$

where: C_D=total drag coefficient

$C_{D,o}$=parasite drag coefficient or zero-lift drag coefficient (it is a constant for a specific aircraft configuration and is not a function of lift)

C_L=lift coefficient

e=Oswald efficiency factor

AR=aspect ratio (b^2/S where b is the wing span and S is the planform area of the wing)

The total drag coefficient is calculated in the UAV data file using relatively common data and the equation shown above. The drag coefficient is a function of the lift coefficient since all of the other parameters are constants for a specific aircraft configuration. Once the drag coefficient is available, it can be used to determine the total aerodynamic force acting on the UAV.

The equations above can now be used to find the total forces acting on the UAV and the thrust required. The lift is calculated using the first equation of 4-5, L=W·cos(climb angle). The lift coefficient is needed as the input into the drag polar look-up table. The lift coefficient is determined using:

$$C_L = \frac{L}{0.5 \cdot \rho \cdot V^2 \cdot S} \tag{4-7}$$

where: C_L=lift coefficient

L=lift (N) (L=W·cos(climb_angle))

ρ=air density (kg/m^3)

V=UAV speed (m/s)

S=planform area of wing (m^2)

The lift coefficient is used as the index into the drag polar look-up table to determine the drag coefficient. The drag coefficient is multiplied by the denominator in Equation 4-7 to calculate the total drag. The total forces acting on the UAV are output to the UAV model block where they are subtracted from the propeller thrust to determine the net thrust. The UAV model uses the inertia of the vehicle and net thrust to calculate the UAV acceleration that is integrated to determine the UAV speed. The speed is integrated to determine the distance traveled. The drag polar is assumed to be the same for all phases of flight which is sufficient for propulsion modeling, even though it is known that the configuration for take-off and landing would change the drag polar.

Chapter 5: Hybrid-Electric UAV Conceptual Design and Sizing

5.1 Introduction

Many hybrid-electric vehicles (HEV), such as those designed for the Future Truck competition, are conversions from stock vehicles. The original power train is removed, and the appropriate components for the hybrid-electric power train are installed. Although the design for the hybrid-electric unmanned aerial vehicle (HEUAV) could take the same approach by using a large scale model aircraft or a manufacturer's UAV, this chapter will briefly step through a conceptual design process to look at the trade-offs that must be considered for sizing the wing and the propulsion system components of a HEUAV. The background provides a fundamental understanding of the requirements for the hybrid-electric propulsion system to enable a better control system design.

The conceptual design approach for aircraft is well described in Anderson, Raymer, Stinton, and Corke [136, 139-141]. Several software packages exist such as ACSYNT [142] and RDS [143] for aircraft sizing and design. Advanced algorithms have been applied to the aircraft conceptual design and sizing problem. Raymer and Crossley use genetic algorithms and evolutionary methods to size aircraft [143]. The techniques permit an optimization of key design variables such as wing design lift coefficient, wing loading, power-to-weight ratio, and aspect ratio. The more advanced genetic algorithms are used instead of the more traditional calculus-based algorithms in order to find a global minimum and avoid local minimums. Holden et al. also use genetic algorithms in a conceptual design process [144]. The literature reveals that defense contractors, universities, and others have devoted much effort to the subject of conceptual design.

Conceptual design is a multi-disciplinary endeavor (aerodynamics, structures, flight controls, weight and balance, cost, etc.), but the focus in this chapter will be the initial sizing of the wing and the propulsion system components in order to scope the problem. A MATLAB routine was written to optimize the size of the wing and the propulsion system components for a conventional high-wing UAV. The HEUAV sizing problem is a constrained optimization formulation. The MATLAB optimization routine uses a sequential quadratic programming method. A quasi-Newton updating method is used at each iteration. The solution of a quadratic programming subproblem is then computed and used in a line search procedure.

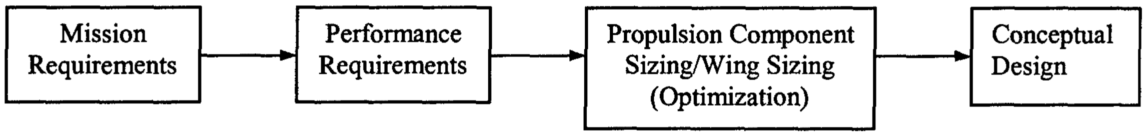


Figure 5-1: Hybrid-Electric Propulsion System Sizing Flowchart

The basic sizing of the wing and the propulsion system components for the HEUAV can be determined using a flowchart as shown in Figure 5-1. To begin, the primary mission requirements of the aircraft must be determined. Once the mission is specified, performance parameters required to satisfy the mission can then be established. The size of the wing and the propulsion system components can then be estimated based on the performance parameters. The result is a conceptual design that would undergo iterations before a preliminary or detailed design is obtained.

Before more details are given about the conceptual design for the HEUAV, several items should be mentioned. First, it is understood that a UAV with a vertical take-off and landing (VTOL) capability would be highly desired for military missions, but a conventional aircraft is used for the purposes of this research. DARPA has considered designing a series hybrid-electric system for a VTOL UAV called the Micro

Air Vehicle [5]. The control algorithms developed in this research could be modified as needed to propel a VTOL aircraft with a hybrid-electric system. Second, the basic assumption that the HEUAV be less than 50 lbs was made for several reasons. The Academy of Model Aeronautics (AMA) safety code requires that a model aircraft be less than 55 lbs at takeoff. If the HEAUV is less than the limit, anyone with an AMA number and trained to fly model aircraft could fly the HEUAV if one was built at an academic institution. Also, a Request for Information (RFI) for small UAVs, written by the Air Force's Aeronautical Systems Center, categorized small UAVs as having a weight of up to 50 lbs. In addition, a UAV in the 25-50 lb range permits one individual to carry the UAV and another to carry the ground control station equipment.

5.2 Mission Requirements

A typical mission for a small UAV is the intelligence, surveillance, and reconnaissance (ISR) mission. A typical flight profile would be to take-off, climb to several thousand feet, cruise for an hour to the location of interest, fly at endurance speed in stealth mode (electric-only) while on station for an hour while conducting ISR, and then return to base (RTB) and land. The HEUAV is sized for this mission. A descent and climb could also be added before and after the time-on-station segment to get a closer look at the area of interest. The typical mission will be used to illustrate the concepts but the sizing process could be easily adapted to other missions.

5.3 Performance Requirements

Performance requirements to satisfy the ISR mission will be used to size the wing and the propulsion system components. The size of the internal combustion engine (ICE), fuel tank, electric motor (EM), and battery pack of the parallel hybrid-electric propulsion system will be based on the performance requirements. Table 5-1 includes a proposed list of the performance requirements.

The hybrid-electric system components are based on the following regimes of operation [13, 145]:

- Take-off power provided by the ICE or the ICE and EM.
- Climbing power provided by the ICE or the ICE and EM.
- Maximum speed power provided by the ICE and EM.
- Cruise power provided by the ICE. A margin is needed to recharge the batteries during charge-sustaining operation.
- Endurance power provided by the EM for stealth operation.
- Missed approaches and emergency power provided by the ICE and EM.

The energy density and mass of the battery pack determines the duration of the "stealth mode." The ICE is primarily sized for cruise, and the EM is primarily sized for the endurance speed. The maximum power from the motor may be needed intermittently during take-off, acceleration, maximum speed, and emergencies. These requirements were analyzed in a typical ISR mission simulation. The flight profile includes take-off, climb, cruise, time-on-station, cruise, descent, approach, and landing.

Table 5-1: HEUAV Performance Requirements

Parameter	Value
Cruise Speed (kts)	45-55 (23.2-28.3 m/s)
Endurance Speed (kts)	20-25 (10.3-12.9 m/s)
Maximum Speed (kts)	60-65 (30.9-33.4 m/s)
Rate-of-Climb (ft/min)	400 (2.0 m/s)
Time for Cruise (Range) (hr)	1
Time at Endurance Speed (hr)	1
Take-off Distance (ft)	80-100 (≈24.4-30.5 m)
Avionics and Payload Power (W)	75
Payload Mass (lbs)	3-6 (≈1.4-2.7 kg)

5.4 Weight Fractions

The weight of the UAV can be expressed using weight fractions as [139]:

$$W_o = \frac{W_{Payload}}{1 - \frac{W_{Fuel}}{W_o} - \frac{W_{Empty}}{W_o}} \qquad \textbf{(5-1)}$$

where W_O is the gross take-off weight. Since the empty weight includes the propulsion system, Equation 5-1 can be modified to separate the propulsion system weight:

$$W_o = \frac{W_{Payload}}{1 - \frac{W_{Fuel}}{W_o} - \left(\frac{W_{Empty} - W_{Propulsion}}{W_o}\right) - \frac{W_{Propulsion}}{W_o}} \qquad \textbf{(5-2)}$$

The term, $(W_{Empty}-W_{Propulsion})/W_O$, will be referred to as the glider weight fraction. Equation 5-2 is used to compare the weight of the original configuration (two-stroke gasoline ICE) to the hybrid-electric configuration. The propulsion system weight fraction for the original configuration includes a larger two-stroke engine, generator, and the propeller. For the parallel hybrid-electric configuration, the propulsion system weight fraction includes the down-sized engine, clutch, batteries, electric motor, and the propeller. The fuel weight fraction, W_{Fuel}/W_O, is determined by computing the amount of fuel needed for each mission segment using estimates and the well known Breguet equation [136]. For the HEUAV, no fuel is used during the endurance mission segment since only electric power is required. The amount of fuel required for the mission is then used to size the fuel tank for the original configuration and the HEUAV configuration.

5.5 Propulsion System Component and Wing Sizing

The sizing of the wing and the propulsion system components are determined from the performance requirements. Due to the potentially large number of variables involved, several were chosen as the key parameters for the sizing process such as the wing loading, aspect ratio, maximum lift coefficient, stall speed, and endurance speed. Several constraints apply to the optimization formulation and will be discussed in detail. The power required at the endurance speed was chosen as the objective function to be minimized. By minimizing the power required at the endurance speed, the weight fraction for the batteries will be minimized while satisfying the constraints and the performance requirements and permitting a feasible payload weight. The logic and concepts involved reveal the trade-offs that must be considered to select the correct sizing of components for the application.

Since the focus of the mission is on the ISR segment, the electric motor and battery weight are first sized to satisfy the one hour of endurance while on station in stealth mode. The power required at the endurance speed is the objective function for the constrained optimization problem. The power required to fly at the endurance speed is given by [136]:

$$PR = \eta_{Prop} \cdot P_{EM} = \sqrt{\frac{2 \cdot W^3 \cdot C_D^2}{\rho \cdot S \cdot C_L^3}} = W \cdot \sqrt{\frac{2 \cdot W \cdot C_D^2}{\rho \cdot S \cdot C_L^3}} = W \cdot \sqrt{\frac{2}{\rho} \cdot \left(\frac{W}{S}\right)} \cdot \frac{4 \cdot C_{D,o}}{\left(3 \cdot C_{D,o} \cdot \pi \cdot e \cdot AR\right)^{0.75}} \qquad \textbf{(5-3)}$$

where: AR=aspect ratio	e=Oswald efficiency factor
C_D=total drag coefficient	ρ=air density (kg/m^3)
$C_{D,o}$=zero-lift drag coefficient	S=wing area (m^2)
C_L=lift coefficient	W=weight of the UAV (N)

The power required is minimized with a small wing loading, W/S, and a large aspect ratio if initial estimates for W, $C_{D,o}$, and e are available. To minimize the power, the result can be thought of as an aircraft that has a large wing area and aspect ratio such as a glider. This type of aircraft would be the most beneficial for sizing the electric system but limits other performance parameters such as the maximum speed. A compromise must be made between this geometry and a smaller wing and aspect ratio. In order to determine the power required from the batteries, it is noted that the power given by Equation 5-3 does not include the propeller efficiency, motor efficiency, and the power required for the payload, avionics, and flight control system.

The parameter $C_L^{3/2} / C_D$ in Equation 5-3 is referred to as the endurance parameter and is found in the literature as the key parameter for solar aircraft or any aircraft with a mission requiring it to fly near or at the endurance speed [146]. Since the ISR mission requires the HEUAV to fly near the endurance speed, the endurance parameter is critical.

An estimate for the zero-lift drag coefficient, $C_{D,o}$, was computed by estimating the type of air flow, wetted area, and the guidelines given in Raymer [139]. For several of the components, the air flow could be laminar but a conservative approach was taken for the estimates so all air flow is considered turbulent. A summary of the estimates is given in Table 5-2. The fineness ratio, FR, and the form factor, FF, are listed. The flat-plate skin friction coefficient, C_f, and the drag coefficient, C_d, for each component are also shown. The reference speed and wing area for the estimates are 10 m/s (≈20 kts) and 1.5 m^2, respectively. The estimates are based on a high-wing conventional aircraft.

Table 5-2: $C_{D,o}$ Estimates for a 30 lb Conventional High-Wing UAV

Component	**L (m)**	**Reynolds Number**	**Boundary Layer**	**S_{wet} (m^2)**	**FR**	**FF**	**C_f**	**C_d**
Fuselage	2.5	1.53x10^6	Turbulent	1.50	9	1.10	4.14x10^3	0.0046
Wing	0.30	1.83x10^5	Turbulent	3.00	NA	1.32	6.27x10^3	0.0166
Horizontal Tail	0.25	1.53 x10^5	Turbulent	0.50	NA	1.25	6.52x10^3	0.0027
Vertical Tail	0.30	1.83 x10^5	Turbulent	0.40	NA	1.25	6.27x10^3	0.0021
Engine Nacelle	0.20	1.22 x10^5	Turbulent	0.15	1	1.35	6.85x10^3	0.0009
Engine Muffler	NA	NA	NA	NA	NA	NA	NA	0.0030
Landing Gear	NA	NA	NA	NA	NA	NA	NA	0.0030
Antennas/Sensors	NA	NA	NA	NA	NA	NA	NA	0.0025
							Total, $C_{D,o}$:	**0.0354**

Several constraints are required to complete the constrained optimization problem. The endurance velocity is given by [136]:

$$V_{\text{Endurance}} = \left(\frac{2}{\rho} \cdot \left(\frac{W}{S} \right) \cdot \sqrt{\frac{1}{3 \cdot C_{D,o} \cdot \pi \cdot e \cdot AR}} \right)^{1/2} \quad \textbf{(5-4)}$$

The endurance velocity is clearly a function of wing loading and the aspect ratio. Squaring each side and rearranging gives the first constraint:

$$\sqrt{AR} = \left(\frac{2}{\rho} \cdot \left(\frac{W}{S} \right) \cdot \sqrt{\frac{1}{3 \cdot C_{D,o} \cdot \pi \cdot e}} \right) \cdot \frac{1}{V^2_{\text{Endurance}}} \quad \textbf{(5-5)}$$

The endurance power required and the endurance velocity are plotted in Figure 5-2 for a 30 lb UAV. The endurance speed and power required increase with increasing wing loading but decrease with increasing aspect ratio. The design point used in the simulations is shown with a small circle.

The second constraint involves the wing loading, stall speed, and the maximum lift coefficient using the following relationship [136]:

$$\frac{W}{S} = \frac{\rho \cdot V^2_{\text{Stall}} \cdot C_{L,\max}}{2} \quad \textbf{(5-6)}$$

The stall speed, for this application and low-speed aircraft, determines the desired wing loading [136]. The landing distance, considered not to be critical for this application, can also determine the wing loading. A speed margin of approximately 3-5 kts is desired between the stall and endurance speeds to account for wind gusts and other disturbances.

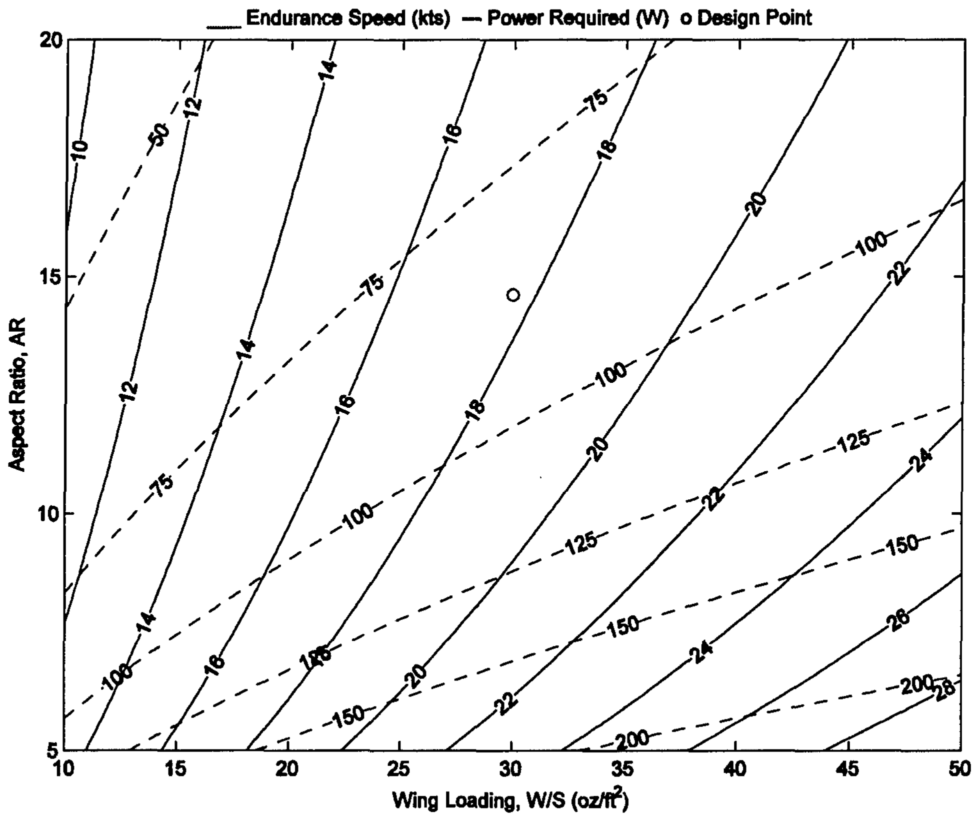


Figure 5-2: Endurance Speed and Power Required for 30 lb UAV

The third constraint determines the size of the gasoline engine. A margin of approximately 125% is used on the ICE so there is extra power from the engine to operate the motor as a generator to recharge the batteries while charge-sustaining and to provide power for the avionics, flight control system, and payload. The expression for the power required during cruise can be expressed as:

$$PR = P_{ICE} \cdot \eta_{Prop} \cdot 0.8 = V_{cruise} \cdot \left(0.5 \cdot \rho \cdot V_{cruise}^2 \cdot S \cdot C_{D,o} + \frac{S}{0.5 \cdot \rho \cdot V_{cruise}^2 \cdot \pi \cdot e \cdot AR} \cdot \left(\frac{W}{S} \right)^2 \right) \quad \textbf{(5-7)}$$

where PR=$V_{cruise} \cdot$TR and TR is the thrust required at cruise speed. Using S=S·W/W, the equation becomes:

$$PR = P_{ICE} \cdot \eta_{Prop} \cdot 0.8 = 0.5 \cdot \rho \cdot V_{cruise}^{3} \cdot W \cdot C_{D,o} \cdot \frac{S}{W} + \frac{W}{0.5 \cdot \rho \cdot V_{cruise} \cdot \pi \cdot e \cdot AR} \cdot \left(\frac{W}{S}\right) \quad \textbf{(5-8)}$$

For the maximum speed, the EM and ICE are both used. The electric motor can tolerate an over-torque for short periods of time, so the expression for the maximum speed is the same as Equation 5-8 except $PR=P_{ICE}\cdot\eta_{Prop}+\text{over\_torque}\cdot\eta_{Prop}\cdot P_{EM}$. Typical values for the over-torque factor are 1.5-2.0.

The objective function and the third constraint form the foundation for a two-point design for the HEUAV. The electric motor and battery pack of the propulsion system are sized based on the endurance speed, and the gasoline engine is sized primarily based on the cruise power requirements. The optimization routine determines the optimum design between these two design points. The design and sizing approach could be adapted to other types of missions.

5.6 Optimization and Conceptual Design Results

The conceptual design results based on the constrained optimization problem led to the following results for an altitude of 5 kft MSL. Table 5-3 includes the optimization routine and conceptual design results. Limits were placed on the variables that were optimized and the results show that the optimization routine used the lowest wing loading available. The size of the wing is directly related to the wing loading. A large wing is desired but other performance parameters must be considered along with the structural, weight, and low-observable requirements. Limits were placed on the maximum lift coefficient in an attempt to obtain results that would permit a standard NACA, Eppler, or Selig airfoil [147, 148] to be used. Other high lift wings could be used such as the low Reynolds number NASA LRN-1-1010 airfoil used in the Navy's low-altitude unmanned research aircraft (LAURA) project [149]. The optimization results either meet or exceed the performance requirements listed in Table 5-1. The endurance speed from the optimization routine is less than the stall speed which is physically not realistic but was permitted to obtain realistic values for the power required at endurance speed. The power required to fly 3-5 kts above the stall speed instead of precisely at the endurance speed is minimal. The power requirements for each mission phase are shown and are used to size the different components. A smaller ICE and less fuel can be used for the HEUAV as compared to the original configuration. The HEUAV requires 25% less fuel than the original configuration for the same mission but with a reduced payload.

Table 5-3: 30 lb UAV Optimization and Conceptual Design Results

Parameter	Value
Optimization Routine Results for HEUAV	
Aspect Ratio	14.6
Wing Loading (N/m^2)	90 (30 oz/ft^2)
Max Lift Coefficient, $C_{L,max}$ (finite wing)	1.25
Oswald Efficiency Factor	0.85
Zero-lift Drag Coefficient	0.036
Stall Speed (m/s)	11.7 (22.7 kts)
Endurance Speed (m/s)	9.1 (17.7 kts)
EM Power Required at Endurance Speed (W)	114
ICE Power Required at Cruise Speed (W)	837
Conceptual Design Results	
Wing Area (m^2)	1.48 (15.9 ft^2)
Wing Span (m)	4.65 (15.3 ft)
Wing Chord (m)	0.32 (12.5 in)
Endurance Parameter	20.4
Max L/D Ratio	16.4
Power Required for Take-Off[1] (W)	503
Power Required for Climb[1] (W)	356
Power Required for Cruise[1] (W)	502
Power Required for Endurance[1] (W)	85
Power Required for Max Speed[1] (W)	852
Nominal Propeller Efficiency (%)	75
Original Fuel Mass (kg)	2.0 (71 oz)
HEUAV Fuel Mass (kg)	1.5 (53 oz)
Original Payload Mass (kg)	3.0 (6.6 lbs)
HEUAV Payload Mass (kg)	1.9 (4.2 lbs)
Original ICE Power Required (W)	1230 (1.7 hp)
HEUAV Battery Mass[2] (kg)	2.2 (4.9 lbs)
HEUAV Battery Storage[2] (Wh)	220

[1]The power required does not include any of the propulsion system inefficiencies or the power required for the avionics and payload.
[2]The battery storage requirement includes the power needed for the payload.

The hybrid-electric propulsion system is a two-point design with the electric system sized for endurance speed and the gasoline engine primarily sized for cruise speed. The endurance speed of approximately 20-25 kts (10.3-12.9 m/s) occurs near the minimum power required of 85 W (see Figure 5-3). The electric motor and battery pack are sized for this speed with additional power for the avionics, payload, and flight control system. The gasoline engine is sized for a cruise speed of 50 kts (25.7 m/s) which requires approximately 500 W, not including the inefficiencies of the propulsion system and the margin required for charge-sustaining operation. The two design points are shown in the figure. Since the electric system is sized for the endurance speed, the weight of the battery pack permits a feasible payload weight.

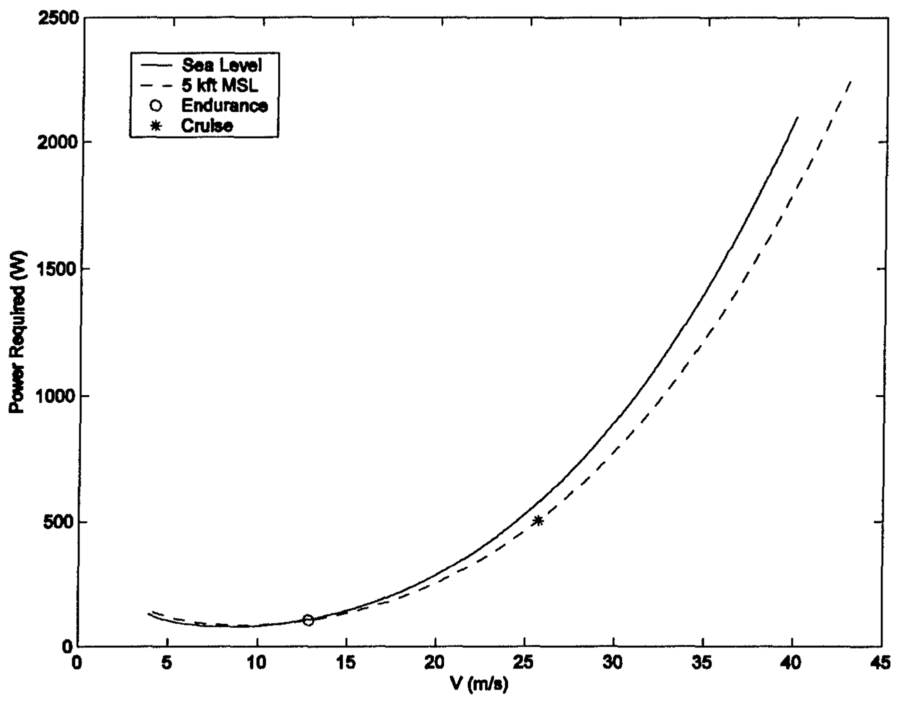


Figure 5-3: Power Required at Sea Level and 5 kft MSL

The weight fractions required for the original and HEUAV configurations are shown in Figure 5-4. The glider weight fraction is estimated from several long endurance UAVs and large scale model airplanes in the same weight class with an empty weight of 0.63. The propulsion weight fraction for the original configuration is 0.12 less than the HEUAV configuration. The fuel weight fraction is approximately 0.11 for the HEUAV and 0.15 for the original configuration. The advantages of the HEUAV configuration must be weighed against the loss in payload mass for a particular mission. The weight fractions for the HEUAV propulsion system components are shown in Figure 5-5. Less fuel is required for the HEUAV, but the battery weight is significant with a weight fraction of 0.16. For comparison, the fuel weight fraction can be as large as 0.35 for UAVs such as the long-endurance Aerosonde UAV.

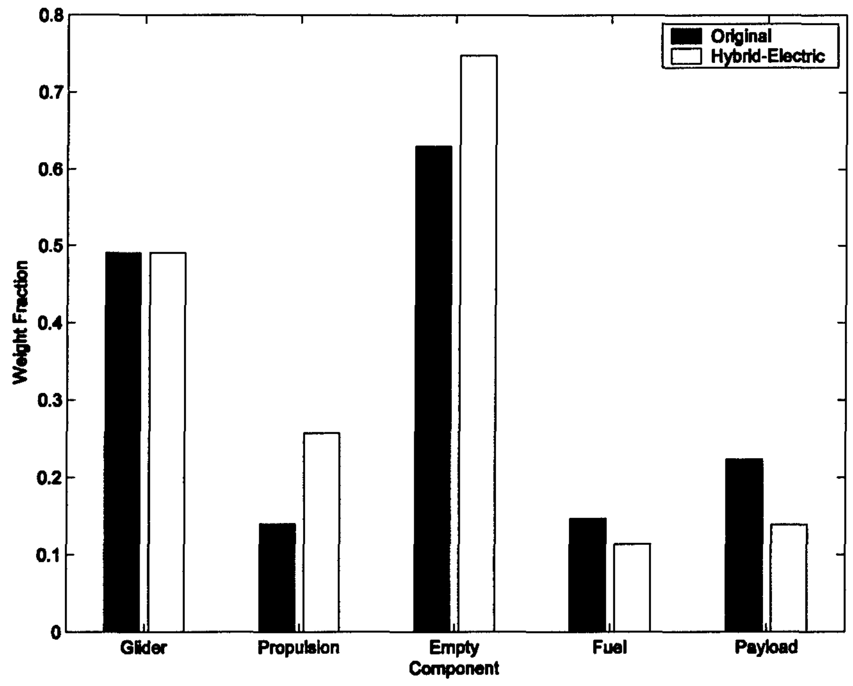


Figure 5-4: Weight Fractions, Normalized to UAV Weight

Off-the-shelf components are matched to the optimization simulation results and are shown in Table 5-4. Since the focus of the research is on the control system for the parallel hybrid-electric propulsion system, nominal design values based on the results from the optimization routine and the components in Table 5-4 are used for the HEUAV propulsion system simulations. The ICE, EM, and battery pack are slightly larger than required which will ensure the power requirements are met for the intended mission. The suppliers for the components are all U.S.-based, so the components are readily available.

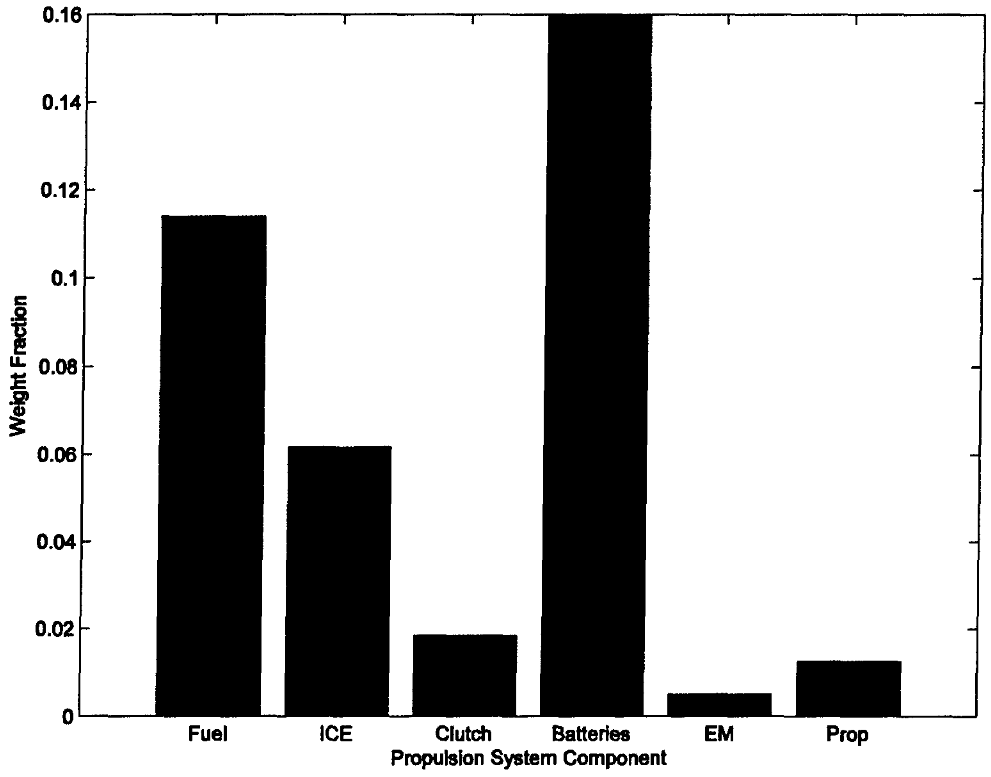


Figure 5-5: Weight Fractions for the HEUAV Propulsion System

Table 5-4: Off-the-Shelf Components for a 30 lb HEUAV

Parameter	Value	Parameter	Value
Airframe	Conventional High-Wing Aircraft	**Battery Pack**	Ultralife UBI-2590 (2 in Parallel)
Mass (kg)	13.6 (30 lbs)	Type	Li-Ion, Rechargeable
Oswald Efficiency Factor	0.85	Mass (kg)	1.44·2=2.88 (6.3 lbs)
Zero-lift Drag Coefficient	0.036	Voltage (V), nominal	14.4
Wing/Airfoil	NACA 23012, E214, S2091, or SD7032	Voltage Range (V)	12.0 to 16.4
$C_{L,max}$	1.25 (3-D, Re=150k)	Capacity, C/2.5 (Ah)	10·2=20
Aspect Ratio	14.6	Energy Density (Wh/kg)	100
Wing Area (m^2)	1.48 (15.9 ft^2)	Recommended Discharge Current (A)	8·2=16 A
Wing Span (m)	4.65 (15.3 ft)	**Electric Motor (3.7:1 Gearbox)**	Aveox 2739/3Y Brushless DC
Wing Chord (m)	0.32 (12.5 in)	Mass (kg)	0.16 (5.6 oz)
Payload		Max Peak Current (A)	30
Original Payload Mass (kg)	2.9 (6.4 lbs)	Max Continuous Current (A)	22
HEUAV Payload Mass (kg)	1.7 (3.7 lbs)	Winding Resistance (Ohms)	0.0817
Original Generator or Battery Pack Mass (kg)	0.5	No Load Current (A)	0.90
Power (W) (include avionics)	75	Speed Constant (rpm/V)	1134
Engine (Two-Stroke, Gas)	First Place Engines	Torque Constant (in-oz/A)	1.19
Displacement (cc)	21 (1.3 in^3) (Original-35 cc)	Motor Constant (in-oz/sqrt(W))	4.28
Mass (kg)	1.13 (2.5 lb)	Maximum Speed (rpm)	50,000
Fuel Tank	54 oz (Original-74 oz)	**Propeller**	Wood, Maple, 18x10
Clutch (Electromagnetic)	RM Hoffman Co.	Mass (kg)	0.17 (6 oz)
Mass (kg)	0.25 (0.55 lbs)	**Horizontal and Vertical Stabilizer**	NACA 0009 Airfoil

A two-stroke gasoline engine was matched to the optimization results since the fuel is available at military installations and because the engines are readily available. A four-stroke engine is also simulated in Chapter 6, but the more efficient engine is heavier and the payload capacity is decreased. Attempts are being made to design small heavy fuel engines for UAVs, so that they use similar fuels as other military vehicles instead of gasoline. A team at Schrick, Inc. has built a 34 kW diesel engine for UAVs with an installed specific weight of 0.7 kg/kW [150]. D-Star Engineering has designed and built a 0.07 kW and a larger 1 kW diesel engine intended for UAVs [151]. For the purposes of the conceptual design in this chapter, typical performance of a two-stroke gasoline engine was used, but the control algorithm approach could be easily adapted to other types of engines.

A conceptual approach to the sizing of the UAV wing and propulsion system components was discussed in this chapter. An optimization problem with constraints was formulated to minimize the power required at or near the endurance speed. The weight of the battery pack was also minimized with this approach. The conceptual design process revealed the trade-offs that must be considered for a hybrid-electric UAV. The optimization and conceptual design results were matched to off-the-shelf components which are used in the simulations. The background explained in this chapter gives a more fundamental understanding of the requirements for the hybrid-electric propulsion system to enable a better control system design.

Chapter 6: CMAC Controller Design and Simulation Results

6.1 Introduction

The conceptual design and sizing of the hybrid-electric unmanned aerial vehicle (HEUAV) explained in the previous chapter revealed the trade-offs that must be considered for the design of the hybrid-electric propulsion system. Using the conceptual design results as a foundation, this chapter covers the design of a CMAC neural network controller and how it approximates the results of an instantaneous optimization algorithm. The optimization of the energy use is achieved by minimizing the instantaneous rate of energy consumption (i.e. using the total power consumption as the objective function to be minimized). The separate nonlinear efficiency maps for the internal combustion engine (ICE), electric motor (EM), and battery pack are used in an off-line instantaneous optimization algorithm to minimize the power consumption by determining the torque split between the ICE and EM during hybrid-electric operation. A control surface for the ICE torque command is generated from the optimization algorithm. A CMAC neural network approximates the optimal control surface and is used in the simulations. Several flight profiles are used in the MATLAB/Simulink HEUAV model (see Chapter 4) to compare the CMAC neural network controller results to the rule-based controller results for the hybrid-electric configuration. Also, the hybrid-electric configuration results are compared to the original configuration (ICE only) simulation results.

6.2 Background

6.2.1 Energy Paths

The energy available in the small HEUAV is either from the gasoline or the electrical energy stored in the battery pack. To provide power to the propeller, the energy can take one of three paths (see Figure 6-1). For Path 1, energy stored within the gasoline is used by the engine to deliver power directly to the propeller. Electrical energy can be delivered directly to the propeller via Path 2. Path 3 uses the engine to recharge the battery pack and the stored electrical energy is delivered to the propeller at a later time. For the HEUAV application, the electrical energy is assumed to be delivered to the propeller at endurance speed (stealth/electric-only mode). The three paths will be referred to during the discussion of the optimization algorithm.

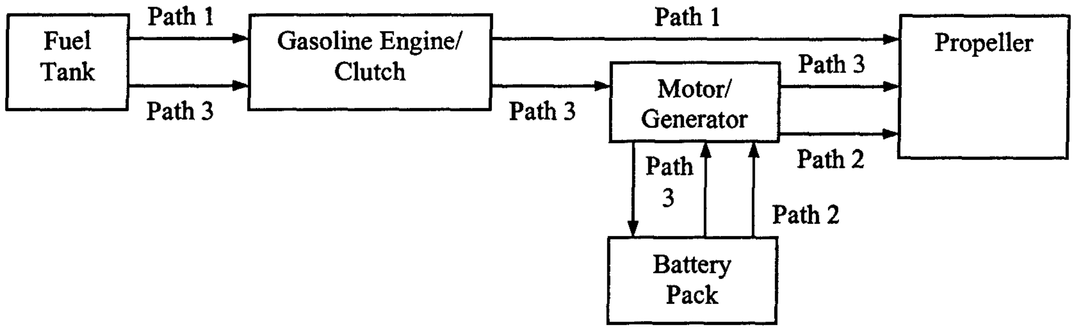


Figure 6-1: Energy Paths Available in the Hybrid-Electric System

6.2.2 Efficiency Maps

The nonlinear efficiency maps for the ICE, EM, and battery pack are used in the off-line optimization algorithm. The maps are stored in tables, and interpolation is used to calculate the efficiency at specific points in the input space (i.e. demanded torque, rotational speed, battery state-of-charge) during the off-line optimization calculations.

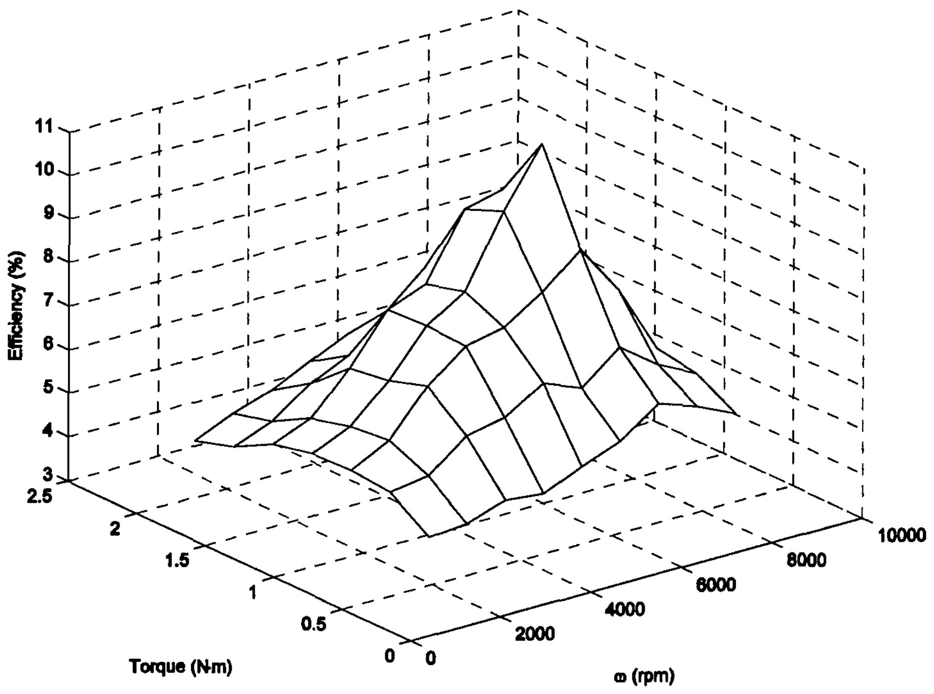


Figure 6-2: Two-Stroke Engine Efficiency Map

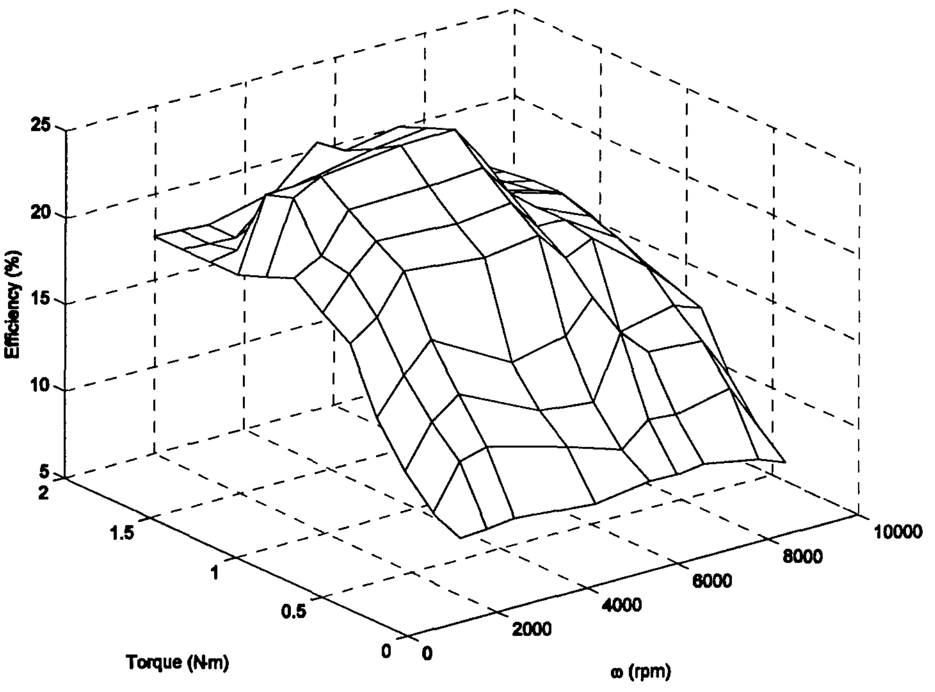


Figure 6-3: Four-Stroke Engine Efficiency Map

The engine efficiency maps are derived from estimates and engine manufacturer dynamometer tests. Estimated fuel consumption for a two-stroke engine was used in the previous chapter for the conceptual design results. This chapter will present results using a two-stroke engine and a more fuel efficient four-stroke engine. The efficiency map for the two-stroke engine was derived for a 21 cc (1.3 in^3) engine using estimates and testing results from First Place Engines [152]. The efficiency map is shown in Figure 6-2. The best fuel consumption is approximately 780 g/kWh or 12% maximum efficiency. The efficiency map for the four-stroke engine was derived from literature for the Honda GX31, 31 cc (1.9 in^3), engine and is shown in Figure 6-3 [153, 154]. The best fuel consumption is approximately 350 g/kWh or a maximum efficiency of 24%.

The electric motor efficiency map is derived from manufacturer data and the following equations:

$$\begin{aligned} T_{EM} &= K_t \cdot (I - I_O) \\ \omega &= K_V \cdot (V - R_m \cdot I) \end{aligned} \qquad \textbf{(6-1)}$$

where: T_{EM}=torque output of the EM (N·m)	ω=rotational speed (rpm)
K_t=torque constant (N·m/A)	K_V=speed constant (rpm/V)
I=current (A)	V=voltage (V)
I_O=no-load current (A)	R_m=winding resistance (Ω)

The EM used in the simulations has a maximum efficiency of 90% (see Figure 6-4).

The battery pack used in the simulations consists of two lithium-ion batteries (Ultralife UBI-2590) in parallel. Data from the manufacturer and a battery model (see Chapter 4) were used to develop an efficiency map for the battery pack with current and the state-of-charge (SOC) as inputs as shown in Figure 6-5. The plot clearly shows the decrease in efficiency with either an increase in discharging or charging current.

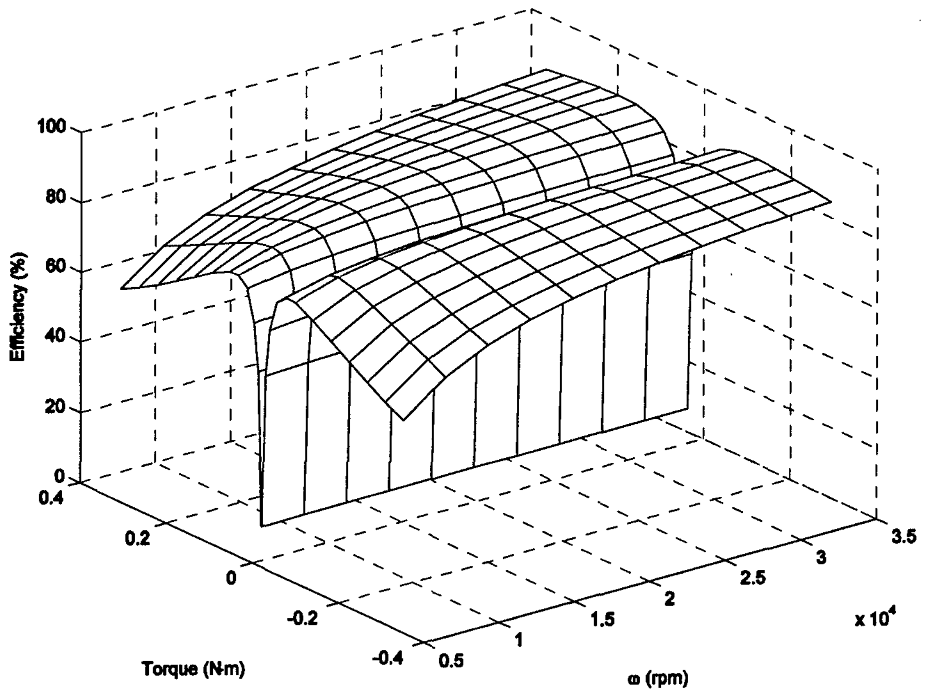


Figure 6-4: Motor Efficiency Map

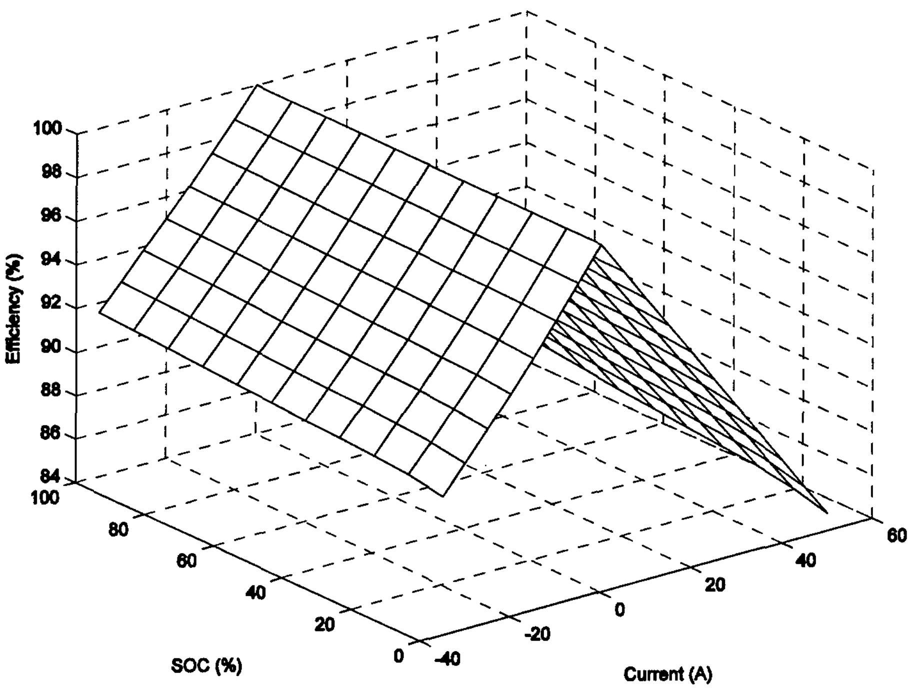


Figure 6-5: Battery Pack Efficiency Map

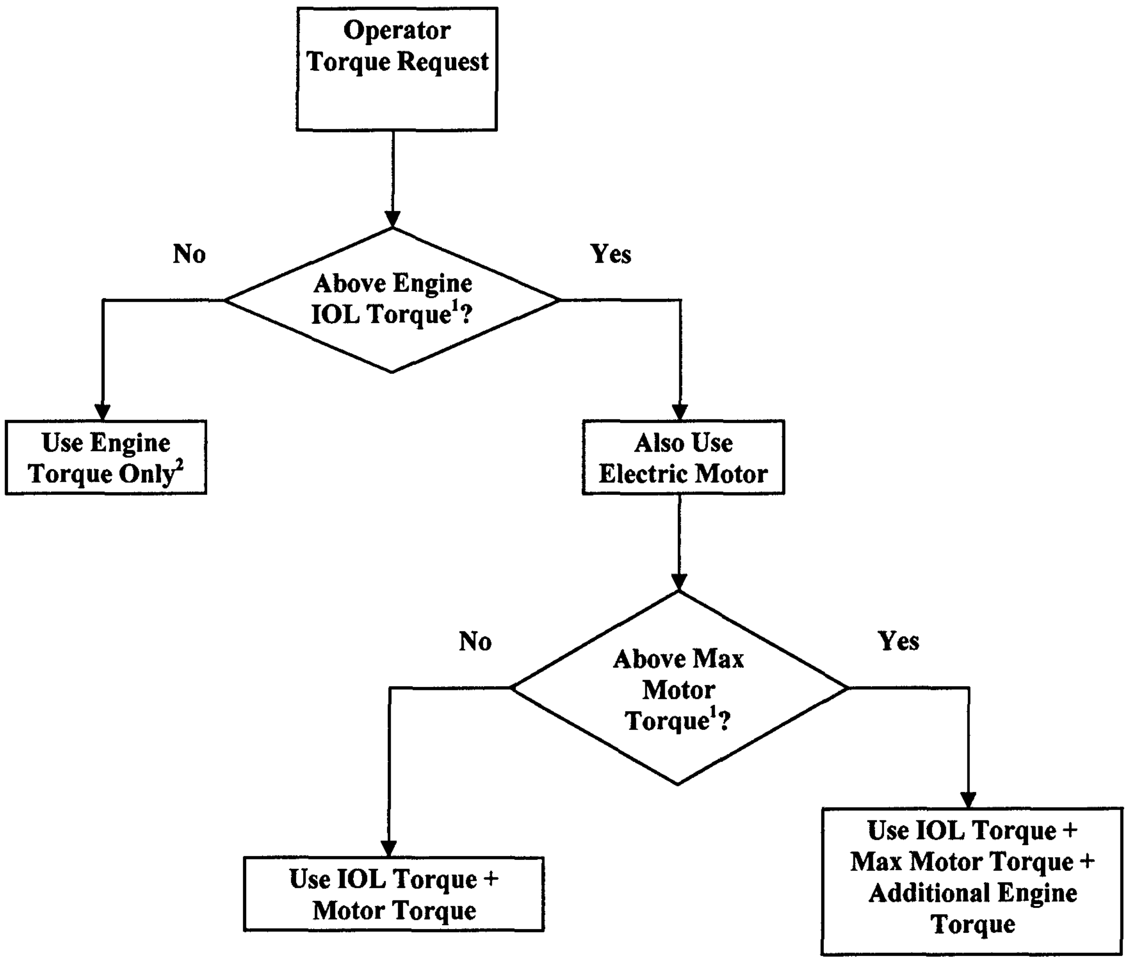
6.3 Optimization Algorithms

The complexity of the optimization algorithm for the hybrid-electric propulsion system controller increased during the development of the controller. The first algorithm duplicates the rule-based algorithm with the rotational speed and demanded torque as the controller inputs to produce a two-input controller. A more complex three-input controller uses the battery SOC as an additional input. A CMAC neural network approximates the output results of the optimization algorithm.

6.3.1 Two-Input Algorithm/Rule-Based Controller

The two-input CMAC controller is designed to duplicate the rule-based controller. The controller has two inputs: demanded torque and rotational speed. The engine is operated on the line of maximum efficiency, the Ideal Operating Line (IOL) [61, 62], unless the demanded torque is less than the IOL torque or if the demanded torque is greater than the combined IOL torque and maximum EM torque. Only the ICE generates torque if the demanded torque is less than the IOL torque. If the demanded torque is greater than the combined IOL torque and the maximum EM torque, then additional torque from the ICE is provided. A flowchart for this algorithm is shown in Figure 6-6.

A plot of the commanded ICE torque generated from the two-input algorithm is shown in Figure 6-7 for the two-stroke engine. The resulting plot is the control surface that the CMAC neural network approximates. The CMAC neural network controller replaces the rule-based controller in the simulations as shown in Figure 1-2. The commanded ICE torque for the four-stroke engine is shown in Figure 6-8. The flat surface represents the IOL torque for a relatively large area of the input space (i.e. rotational speed and total desired torque).


Operator Torque Request
Above Engine IOL Torque[1]?
No
Yes
Use Engine Torque Only[2]
Also Use Electric Motor
Above Max Motor Torque[1]?
No
Yes
Use IOL Torque + Motor Torque
Use IOL Torque + Max Motor Torque + Additional Engine Torque

[1]Requires a Table Look-Up
[2]If Torque Request < 0 N·m, then Engine Torque = 0 N·m

Figure 6-6: Two-Input Algorithm/Rule-Based Controller

The flowchart shown in Figure 6-6 does not include any recharging, so it is considered a charge-depletion (CD) strategy. If the mission requirements cannot be met with CD only, then a charge-sustaining (CS) algorithm is used. The CS algorithm is based on the expected length of the mission and the battery SOC. A proportional-derivative (PD) controller, with the SOC as an input, determines the amount of recharging required (see Figure 4-10).

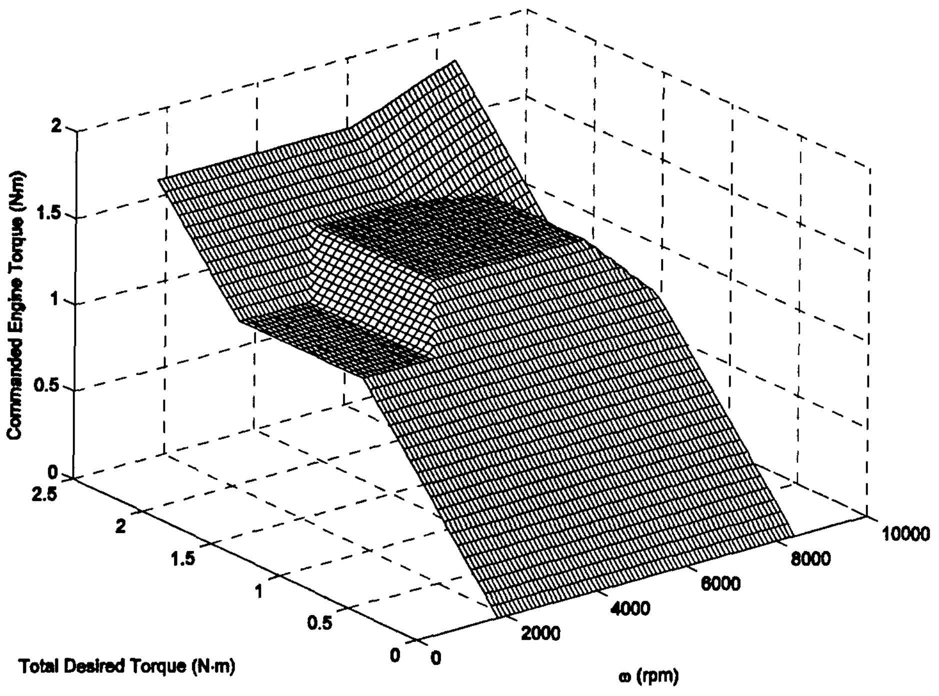


Figure 6-7: Engine Torque Control Surface, Two-Stroke, Two-Input Algorithm

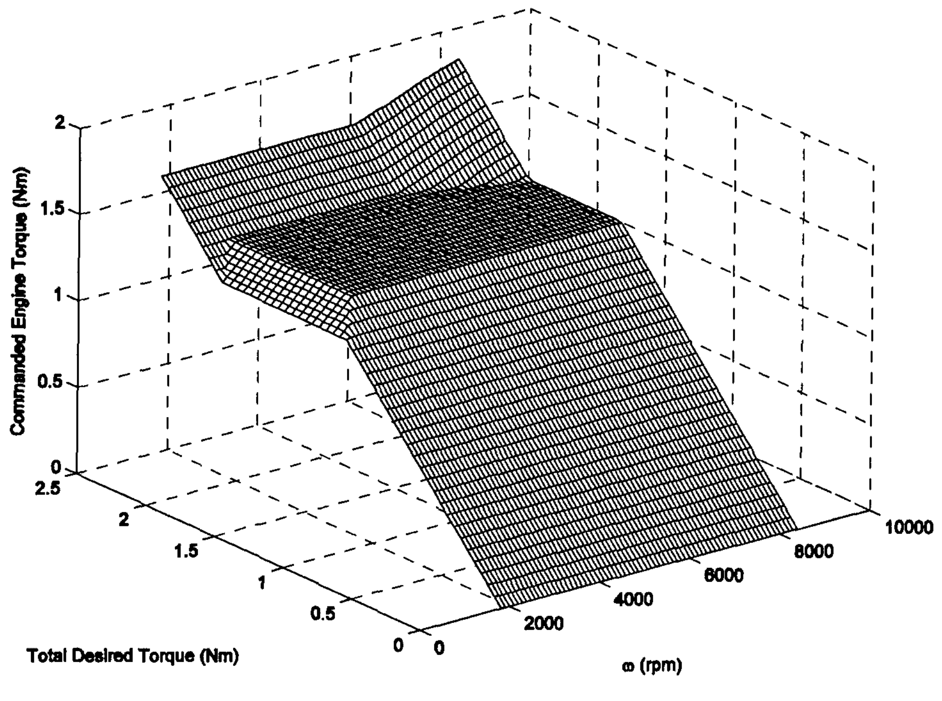


Figure 6-8: Engine Torque Control Surface, Four-Stroke, Two-Input Algorithm

6.3.2 Three-Input Algorithm

The three-input controller uses the battery SOC as an input in addition to the demanded torque and rotational speed. The algorithm minimizes the total power consumption of the engine and the motor:

$$J = P_{ICE} + \alpha \cdot P_{EM} + \beta \cdot P_{EM\_recharge} \qquad \textbf{(6-2)}$$

P_{ICE} is the power consumption equivalent (33.44 kWh/gal of gasoline) of the ICE to rotate the propeller (Path 1). P_{EM} is the electrical power consumption of the EM (Path 2), whereas $P_{EM\_recharge}$ is the power consumption equivalent for the ICE to operate the EM as a generator to recharge the batteries (Path 3). The weighting factors, α and β, penalize the amount of electricity use and the amount of recharging, respectively, and are mission dependent. If the torque of the engine is greater than the demanded torque, then the motor is used as a generator to recharge the batteries.

The flowchart for the optimization algorithm is shown in Figure 6-9. The logic and calculations involved such as the table look-ups would be excessive for an embedded microcontroller [27]. For the appropriate branches of the flowchart, the objective function is calculated in the off-line optimization by stepping the engine torque by 0.02 N·m increments to determine which torque split will minimize the power consumption. The values determined for α and β depend on the type of mission. If the mission is a relatively short mission, values that encourage CD are used, but if a longer mission is required, different values to produce a CS control surface are used. For both cases, more charging can be produced as the battery SOC decreases. Examples for the two-stroke and four-stroke engines for both CS and CD control surfaces will be given.

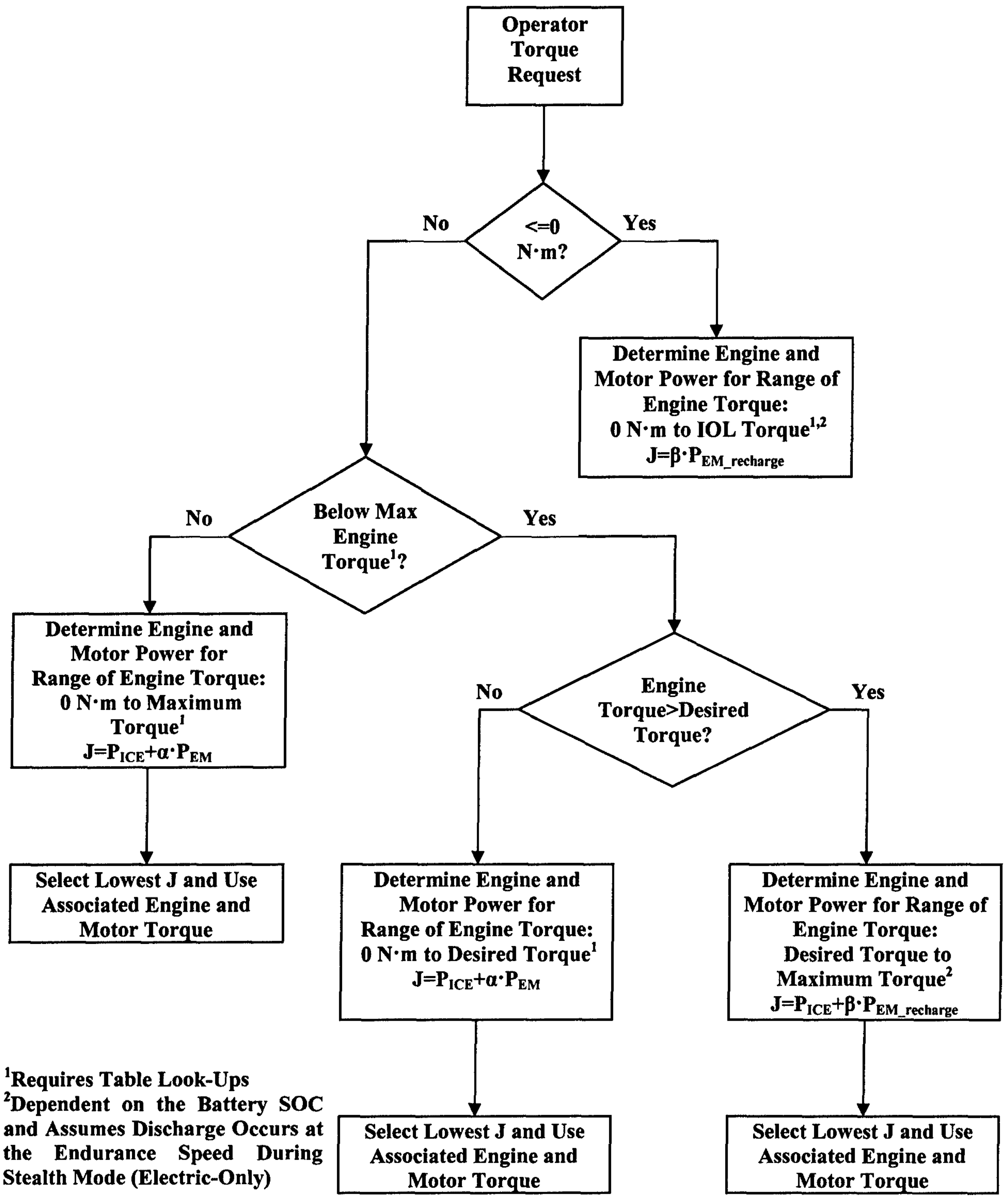

Operator Torque Request
<=0 N·m?
No
Yes
Determine Engine and Motor Power for Range of Engine Torque: 0 N·m to IOL Torque^{1,2} $J=\beta\cdot P_{EM\_recharge}$
Below Max Engine Torque^{1}?
No
Yes
Determine Engine and Motor Power for Range of Engine Torque: 0 N·m to Maximum Torque^{1} $J=P_{ICE}+\alpha\cdot P_{EM}$
Engine Torque>Desired Torque?
No
Yes
Select Lowest J and Use Associated Engine and Motor Torque
Determine Engine and Motor Power for Range of Engine Torque: 0 N·m to Desired Torque^{1} $J=P_{ICE}+\alpha\cdot P_{EM}$
Determine Engine and Motor Power for Range of Engine Torque: Desired Torque to Maximum Torque^{2} $J=P_{ICE}+\beta\cdot P_{EM\_recharge}$
Select Lowest J and Use Associated Engine and Motor Torque
Select Lowest J and Use Associated Engine and Motor Torque
^{1}Requires Table Look-Ups
^{2}Dependent on the Battery SOC and Assumes Discharge Occurs at the Endurance Speed During Stealth Mode (Electric-Only)

Figure 6-9: Three-Input Optimization Algorithm

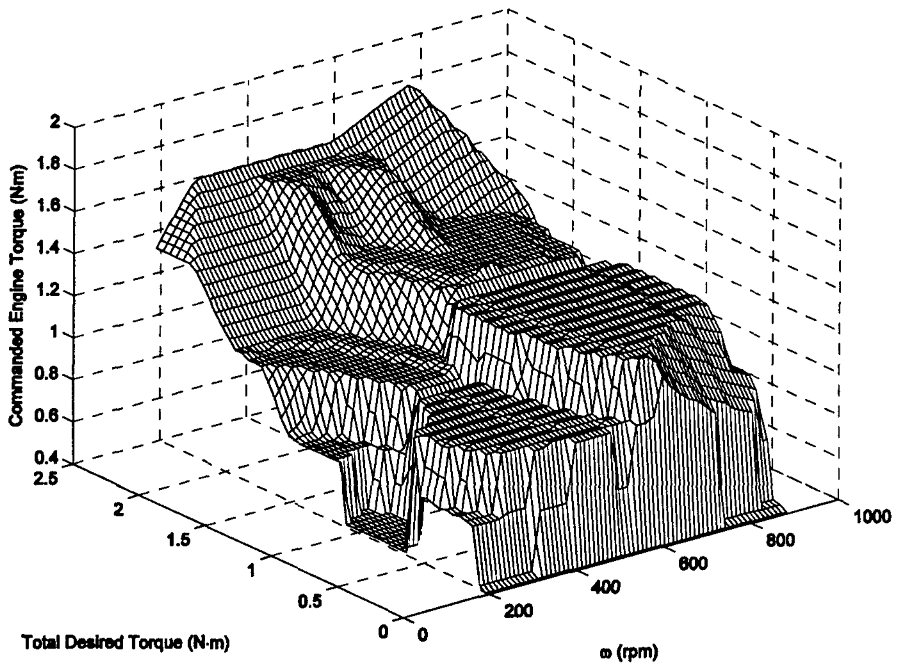


Figure 6-10: Engine Torque Control Surface, Two-Stroke, Three-Input Algorithm, Charge-Sustaining, SOC=100%

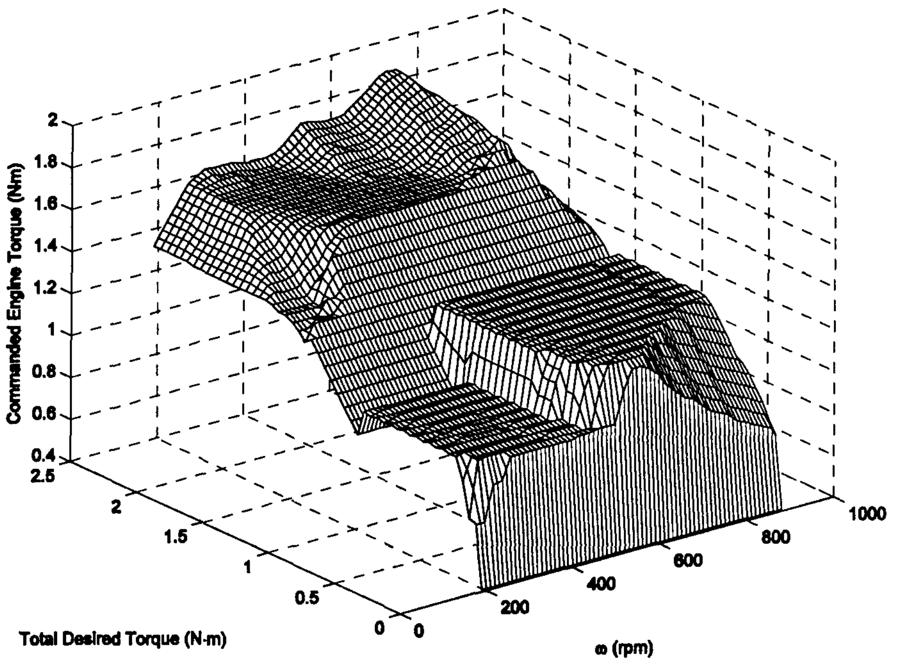


Figure 6-11: Engine Torque Control Surface, Two-Stroke, Three-Input Algorithm, Charge-Sustaining, SOC=25%

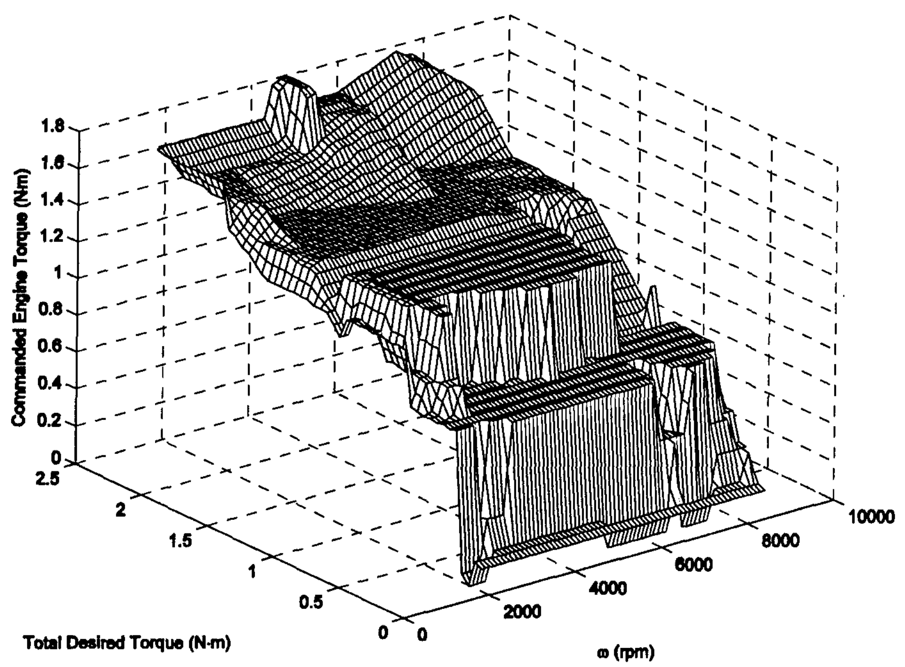


Figure 6-12: Engine Torque Control Surface, Four-Stroke, Three-Input Algorithm, Charge-Sustaining, SOC=100%

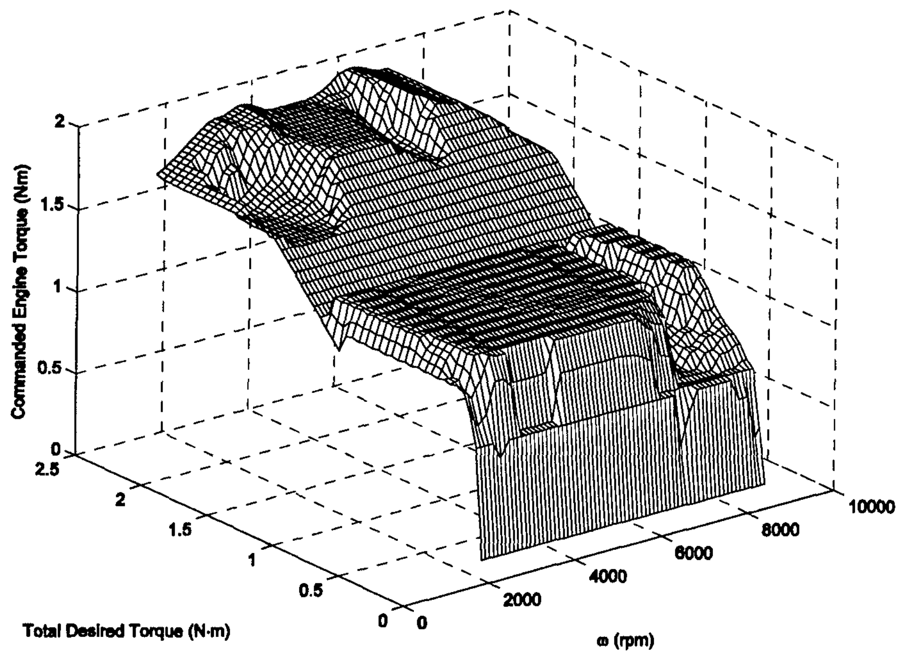


Figure 6-13: Engine Torque Control Surface, Four-Stroke, Three-Input Algorithm, Charge-Sustaining, SOC=25%

Examples of the CS control surfaces for the two-stroke and four-stroke engines are shown in Figures 6-10 through 6-13. For the two-stroke engine, the weighting values are α =10.0+10.0·(1-SOC) and β=0.05·SOC. The weighting factors are α =4.0+4.0·(1-SOC) and β=0.3·SOC for the more efficient four-stroke engine. The values were determined to allow the UAV to complete a three-hour ISR mission. More recharging is performed as the SOC decreases while still minimizing the objective function. The optimization algorithm produces control surfaces that cause the engine to operate near or at the IOL, especially for a low battery SOC.

Examples for the two-stroke and four-stroke engines to produce charge-depletion control surfaces are shown in Figures 6-14 through 6-17. For the two-stroke engine, the α and β values are set equal to 6.5+1.0·(1-SOC) and 5.0·SOC, respectively. The weighting factors are α =3.0+1.0·(1-SOC) and β=2.0·SOC for the more efficient four-stroke engine. The values were determined to allow the UAV to complete a one-hour ISR mission. These values emphasize CD and the engine is not operated on the IOL as often. The CD surfaces cause more electrical energy to be used. For low power settings, only the engine or only electrical energy is used to meet the torque demand. The charge-depletion surfaces allow the HEUAV to complete a shorter one-hour mission.

The torque control surfaces shown in Figures 6-7 and 6-8, and 6-10 through 6-17 can be thought of as control surfaces that can be stored in a look-up table. However, a look-up table requires a memory location for every value in the input space or fewer values and more interpolation. The CMAC neural network approximations to these surfaces will require less memory storage and are used in the HEUAV simulations.

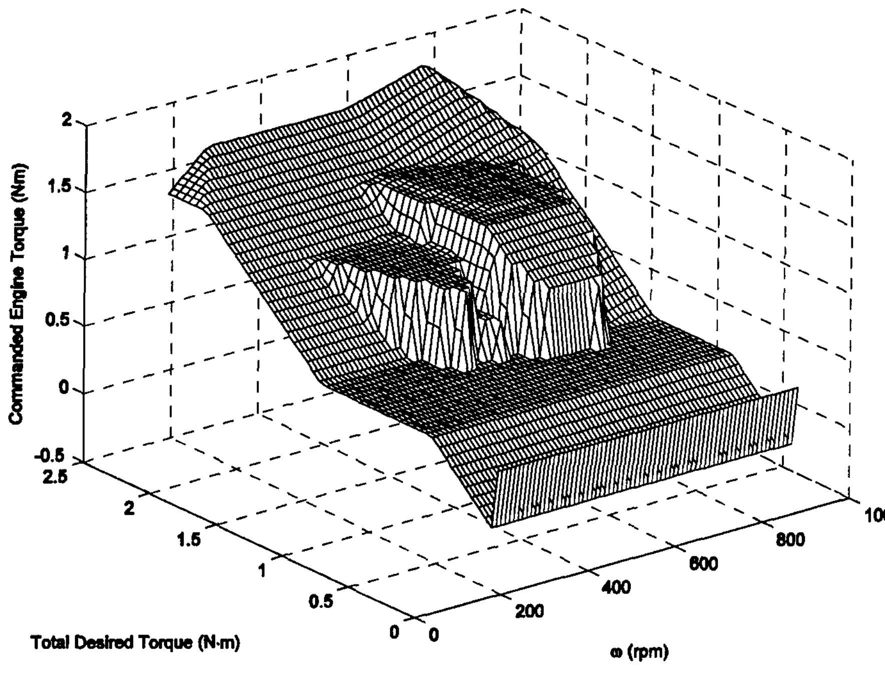


Figure 6-14: Engine Torque Control Surface, Two-Stroke, Three-Input Algorithm, Charge-Depletion, SOC=100%

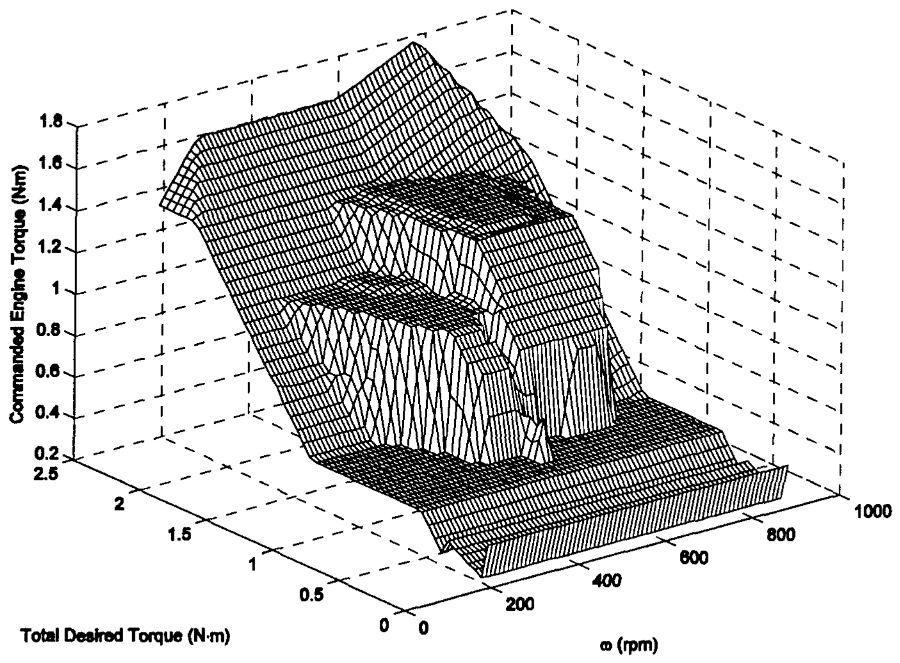


Figure 6-15: Engine Torque Control Surface, Two-Stroke, Three-Input Algorithm, Charge-Depletion, SOC=25%

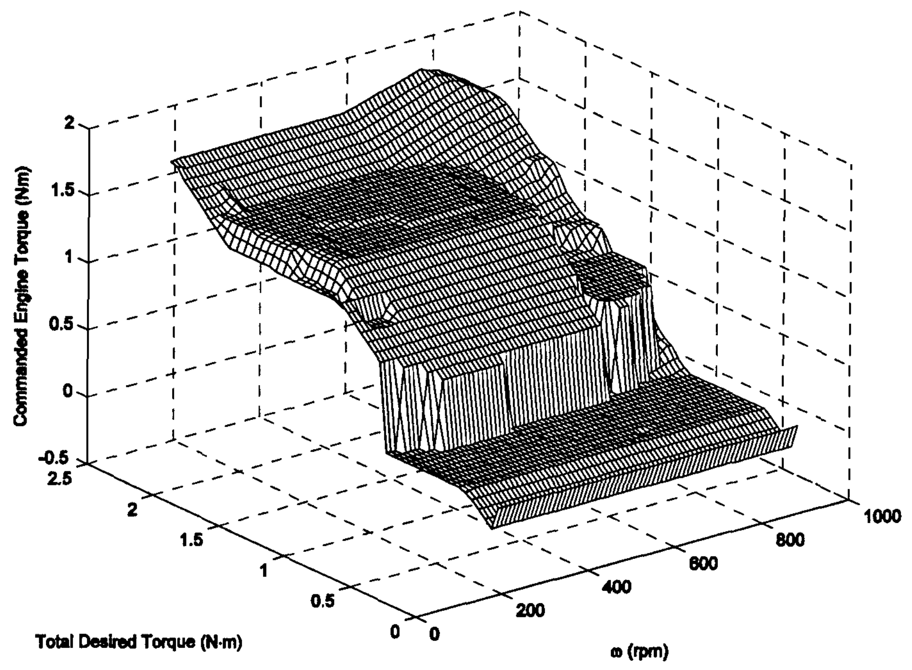


Figure 6-16: Engine Torque Control Surface, Four-Stroke, Three-Input Algorithm, Charge-Depletion, SOC=100%

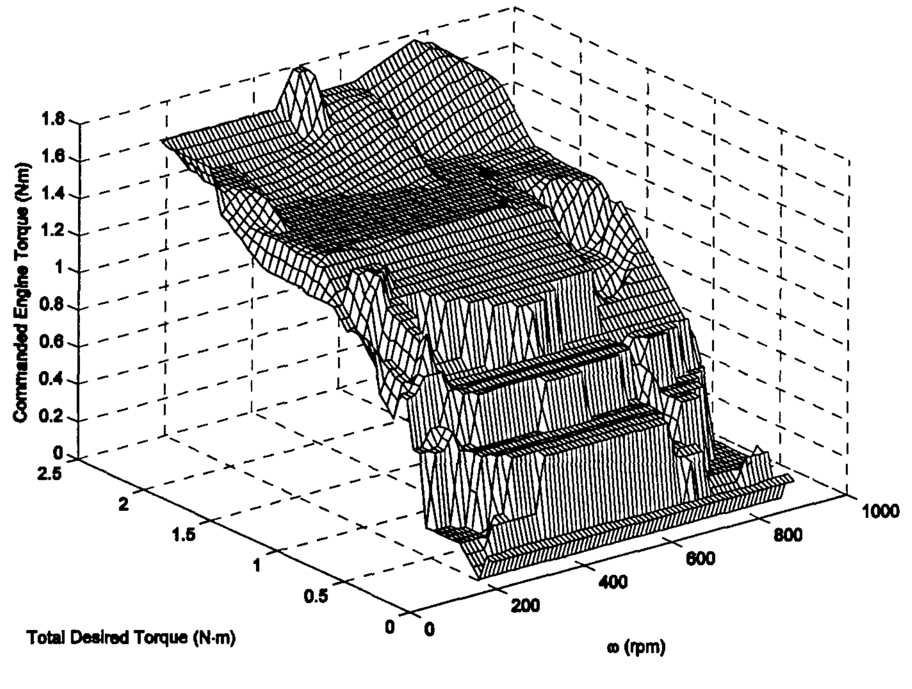


Figure 6-17: Engine Torque Control Surface, Four-Stroke, Three-Input Algorithm, Charge-Depletion, SOC=25%

6.4 CMAC Neural Network Controller Simulink Block

The CMAC neural network Simulink block (see Figure 6-18) replaces the rule-based controller in the HEUAV MATLAB/Simulink model as described in Chapter 4. Parameters such as the propeller speed, demanded torque, and the battery SOC are used as inputs. The index for each input is then output from the block. Once the location in the input space is determined from the inputs, an S-function is used to determine the output value of the CMAC neural network. The values of the CMAC neural network approximations are used to determine the engine torque command. The electric motor torque command is determined by subtracting the engine torque from the desired torque. The output commands are used in the model in the same way as the commands from the rule-based controller.

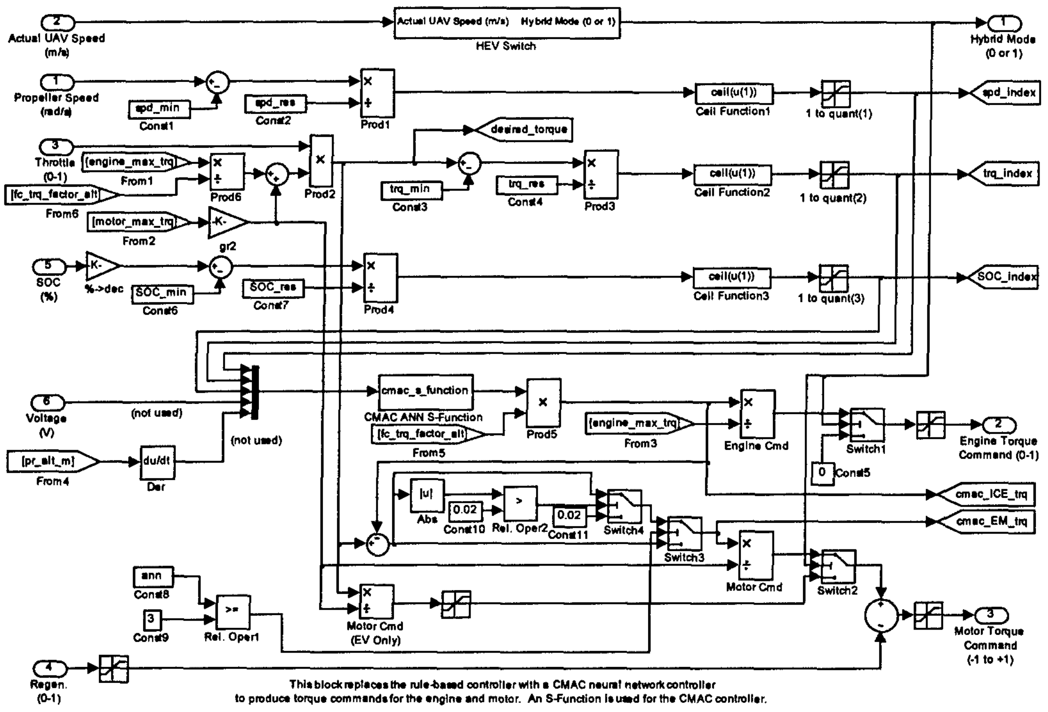


Figure 6-18: CMAC Controller Simulink Block

6.5 CMAC Controller Approximations

To implement the CMAC controller, various parameters were chosen such as the inputs, quantization widths, generalization parameter, and learning rate. The parameters chosen determine the accuracy of the CMAC approximation for the control surface. Brown and Harris state that "that most reasonable choices give acceptable results" [127]. A summary of the CMAC parameters for each HEUAV propulsion system controller are given in Table 6-1.

The CMAC controller inputs that were chosen are measurable with relatively inexpensive commercial off-the-shelf (COTS) components. These inputs include rotational speed, demanded torque, and the battery SOC. The torque output of the engine and electric motor could be useful, but experience with hybrid-electric automobiles illustrate that it is reasonable to use steady-state performance maps for the propulsion system components.

Table 6-1: Parameter Summary for the HEUAV CMAC ANN Controllers

Parameter	Two-Input Controller	Three-Input Controller
Output	Commanded Engine Torque	Commanded Engine Torque
Inputs	Rotational Speed Demanded Torque	Rotational Speed Demanded Torque Battery State-of-Charge
Input Ranges	Speed: 190-880 rad/s Torque: 0-2.5 N·m	Speed: 190-880 rad/s Torque: 0-2.5 N·m SOC: 2-100%
Input Resolution	Speed: 10 rad/s Torque: 0.05 N·m	Speed: 10 rad/s Torque: 0.05 N·m SOC: 2%
Number of Input Cells[1]	70·51=3,570	70·51·50=178,500
Generalization Factor[2]	3,5,8,10,13,15,18	3,7,9,14,19, 25
Learning Rate	0.05	0.05
Training Iterations	250	150-200

[1]Entries required for a Look-Up Table (LUT)

[2]Most of the generalization factors used correspond to displacement vectors that provide a CMAC structure of good quality [127]

6.5.1 Two-Input CMAC Controller Approximations

The function approximations for the two-input controller illustrate how the CMAC neural network approximates the engine torque control surface. The two-input controller duplicates the rule-based controller. The more complex three-input CMAC controller improves on the two-input/rule-based controller.

Two-Stroke Engine Approximations: A summary of the CMAC approximation results using the two-stroke engine are shown in Table 6-2. The RMS error increases with an increase in the generalization parameter. The memory savings is a factor of 10.98 (one order of magnitude) as compared to a look-up table (LUT) for L=18.

Table 6-2: Summary of the CMAC Approximation Results for the Two-Input Controller, Charge-Depletion, Two-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$3.47\cdot10^{-3}$	(1,1)	1255	2.84
5	$3.71\cdot10^{-3}$	(1,2)	803	4.45
8	$5.05\cdot10^{-3}$	(1,3)	551	6.48
10	$6.47\cdot10^{-3}$	(1,3)	468	7.63
13	$7.92\cdot10^{-3}$	(1,5)	393	9.08
15	$9.33\cdot10^{-3}$	(1,4)	361	9.88
18	$9.87\cdot10^{-3}$	(1,5)	325	10.98

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

The CMAC approximation (L=3) to the rule-based controller is shown in Figure 6-19. If the generalization factor is increased to 18, the modeling error of the CMAC approximation gets worse as shown in Figure 6-20. The generalization factors used are considered “good quality” due to the displacement vector [127]. If L is increased above ≈20, the RMS error increases and acceptable results are not produced. This can be attributed to a CMAC structure that is not well defined. The RMS errors during training for several cases in Table 6-2 are shown in Figure 6-21. As the generalization factor increases, the initial convergence is faster, but the resulting RMS error is worse.

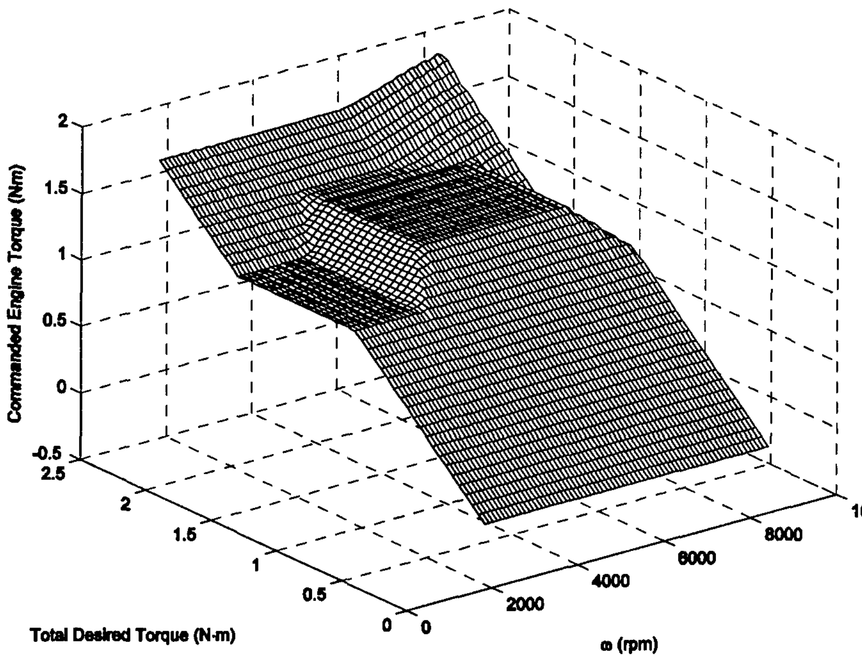


Figure 6-19: CMAC Approximation (L=3) for Engine Torque Control Surface (see Figure 6-7), Two-Stroke, Two-Input Controller

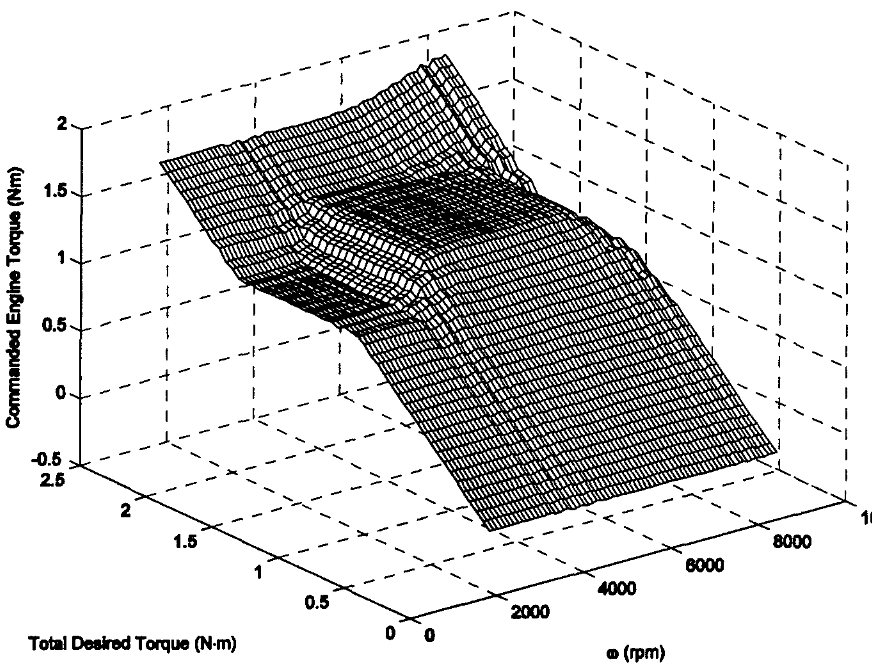


Figure 6-20: CMAC Approximation (L=18) for Engine Torque Control Surface (see Figure 6-7), Two-Stroke, Two-Input Controller

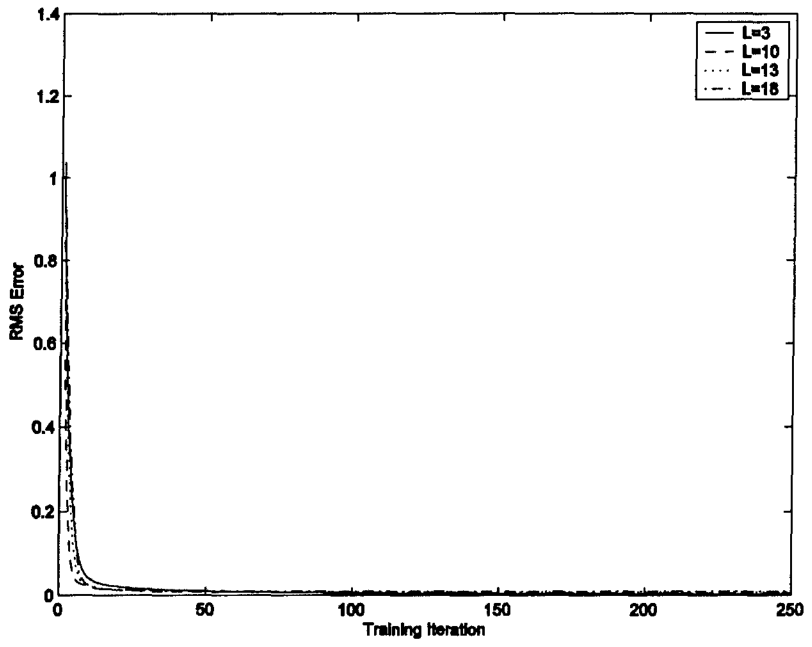


Figure 6-21: RMS Error, Two-Stroke, Two-Input Controller

Four-Stroke Engine Approximations: A summary of the CMAC approximation results for the two-input controller using the four-stroke engine are shown in Table 6-3. The CMAC approximation (L=18) for the two-input control surface is shown in Figure 6-22. For the four-stroke engine controller surface, the error is still minimal with L=18. The RMS errors during training for several cases in Table 6-3 are shown in Figure 6-23. The number of training iterations was 250 although only 100 are shown in the plot.

Table 6-3: Summary of the CMAC Approximation Results for the Two-Input Controller, Charge-Depletion, Four-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$3.62 \cdot 10^{-3}$	(1,1)	1255	2.84
5	$3.19 \cdot 10^{-3}$	(1,2)	803	4.45
8	$3.54 \cdot 10^{-3}$	(1,3)	551	6.48
10	$3.73 \cdot 10^{-3}$	(1,3)	468	7.63
13	$4.67 \cdot 10^{-3}$	(1,5)	393	9.08
15	$5.37 \cdot 10^{-3}$	(1,4)	361	9.89
18	$5.05 \cdot 10^{-3}$	(1,5)	325	10.98

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

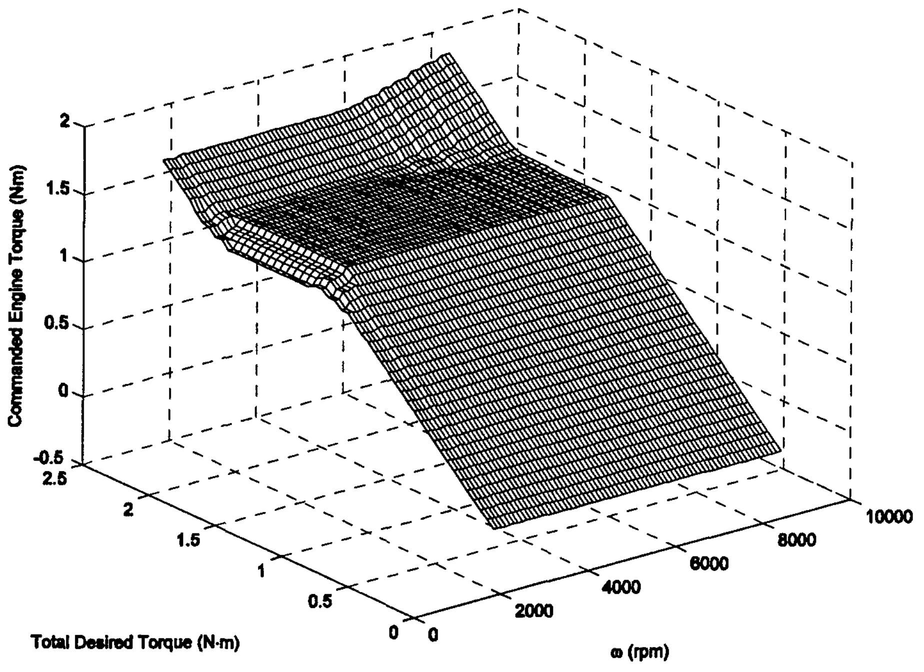


Figure 6-22: CMAC Approximation (L=18) for Engine Torque Control Surface (see Figure 6-8), Four-Stroke, Two-Input Controller

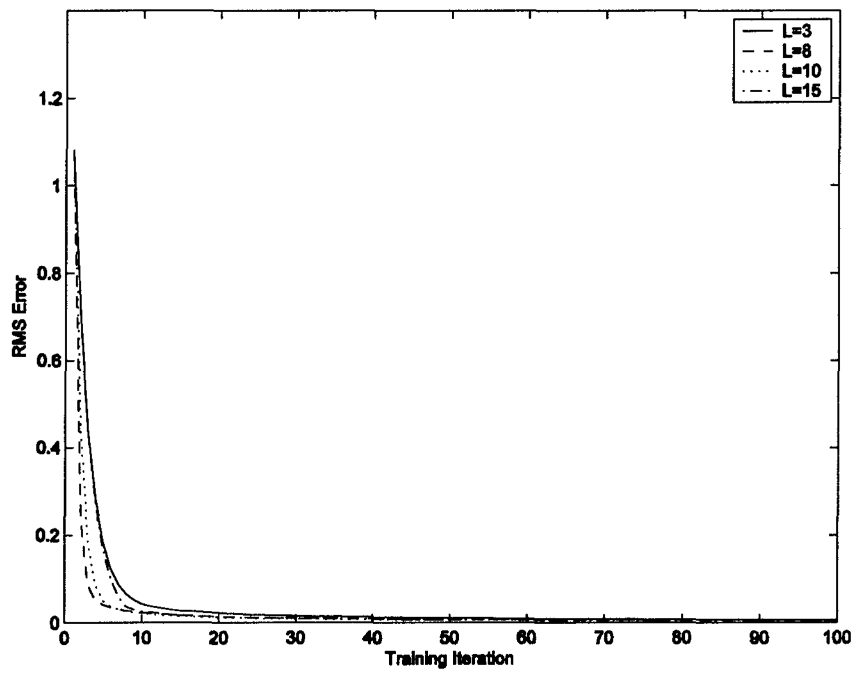


Figure 6-23: RMS Error, Four-Stroke, Two-Input Controller

6.5.2 Three-Input CMAC Controller Approximations

The control surface approximations for the three-input controller reveal how the CMAC neural network can accurately approximate a four-dimensional surface. The CMAC ANN approximates the results of the instantaneous optimization algorithm. The optimization algorithm is an improvement over the rule-based controller. Different surfaces are generated for CS and CD operation. The CS surfaces are used for a three-hour ISR mission. The CD control surfaces are used for a shorter one-hour ISR mission.

Two-Stroke Engine, Charge-Sustaining Approximations: A summary of the approximation results are shown in Table 6-4 for the two-stroke engine for charge-sustaining operation. CS weighting factors of $\alpha=10.0+10\cdot(1\text{-SOC})$ and $\beta=0.05\cdot\text{SOC}$ permit the HEUAV to complete a three-hour mission. The runs for L=3, 7, and 9 used 200 training iterations, and the other two runs used 150 iterations. The CMAC approximations for L=14 and 19 save on at least two orders of magnitude as compared to a LUT.

Table 6-4: Summary of the CMAC Approximation Results for the Three-Input Controller, Charge-Sustaining, Two-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$1.31\cdot10^{-2}$	(1,1,1)	21,767	8.20
7	$2.41\cdot10^{-2}$	(1,2,3)	4,888	36.5
9	$2.90\cdot10^{-2}$	(1,2,4)	3,266	54.7
14	$4.37\cdot10^{-2}$	(1,3,5)	1,696	105.2
19	$5.82\cdot10^{-2}$	(1,3,7)	1,137	157.0

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

The CMAC approximations for L=3 and L=14 for a SOC of 100% and 25% are shown in Figures 6-24 through 6-27. The CS surfaces for L=3, 7, 9, and 14 produce acceptable results. A generalization factor of L=19 does not since the approximation causes too much recharging due to the modeling errors.

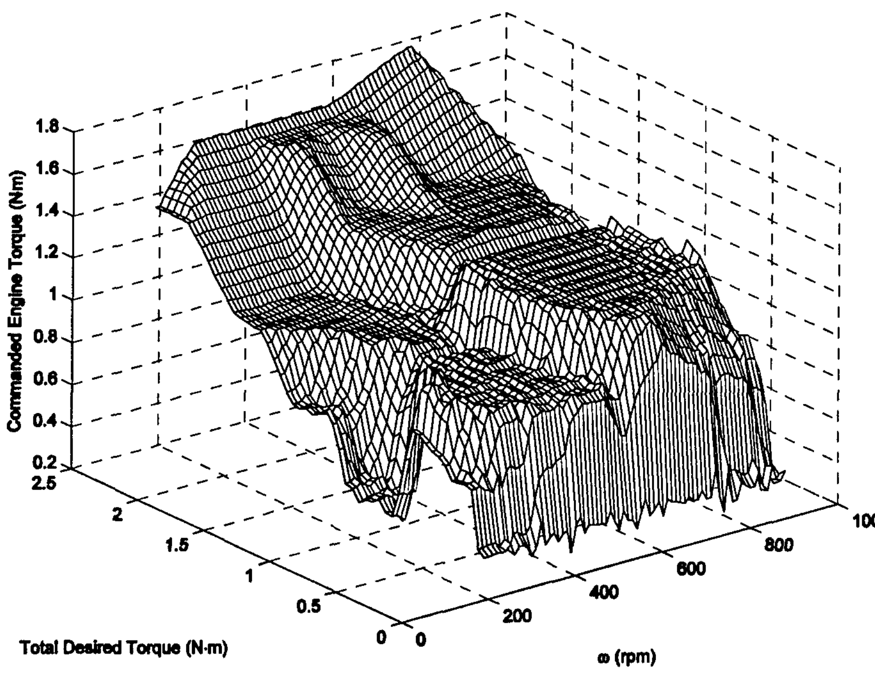


Figure 6-24: CMAC Approximation (L=3) for Engine Torque Control Surface (see Figure 6-10), Two-Stroke, Three-Input Controller, Charge-Sustaining, SOC=100%

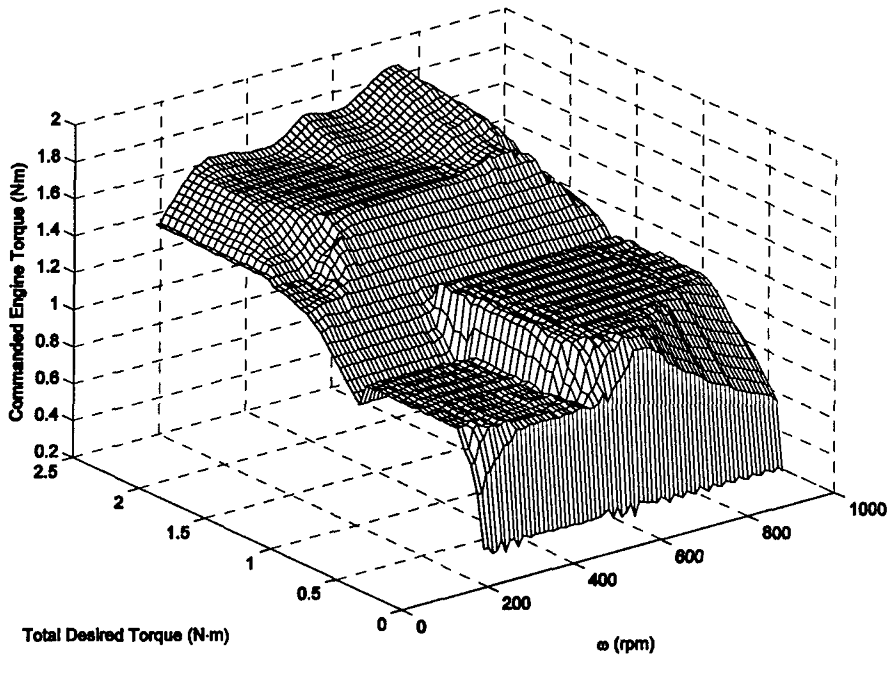


Figure 6-25: CMAC Approximation (L=3) for Engine Torque Control Surface (see Figure 6-11), Two-Stroke, Three-Input Controller, Charge-Sustaining, SOC=25%

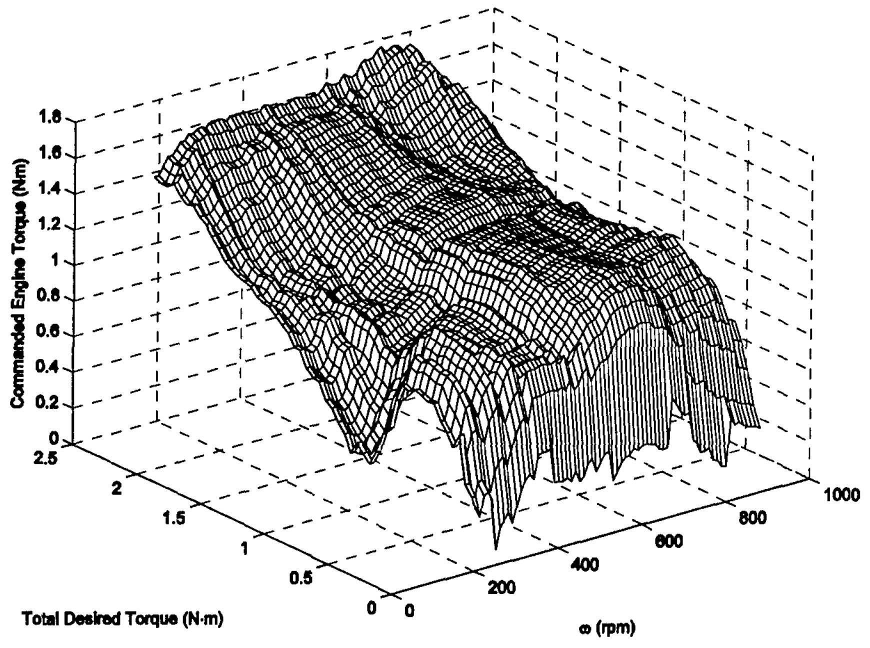


Figure 6-26: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-10), Two-Stroke, Three-Input Controller, Charge-Sustaining, SOC=100%

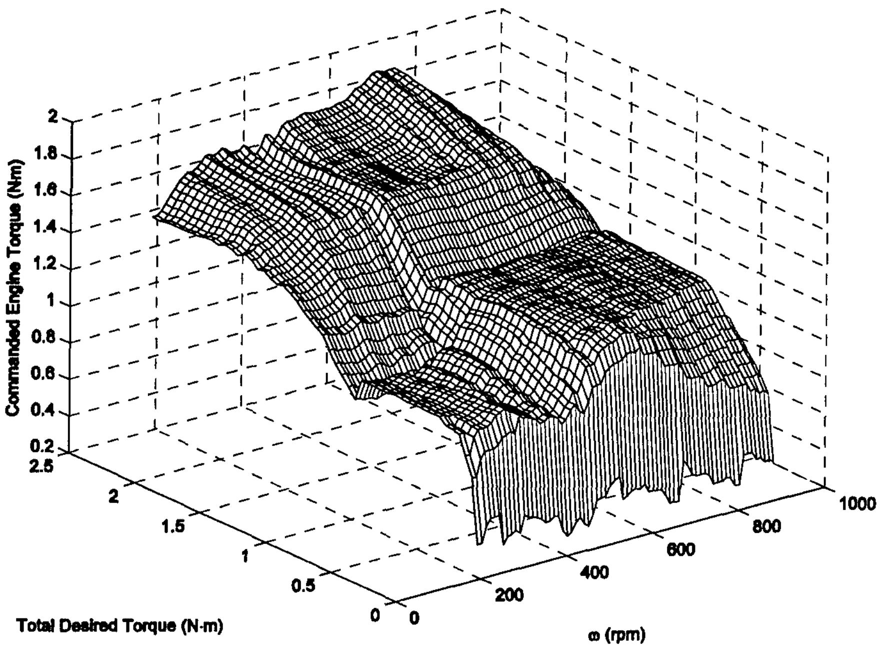


Figure 6-27: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-11), Two-Stroke, Three-Input Controller, Charge-Sustaining, SOC=25%

Four-Stroke Engine, Charge-Sustaining Approximations: A summary of the CMAC approximation results for the three-input controller using the four-stroke engine are shown in Table 6-5. Charge-sustaining values for α and β were determined that permit the HEUAV to complete a three-hour ISR mission. The runs for L=3, 7, and 9 used 200 training iterations, and the other runs used 150 iterations. The CMAC approximation (L=14) saves on two orders of magnitude as compared to a LUT.

Table 6-5: Summary of the CMAC Approximation Results for the Three-Input Controller, Charge-Sustaining, Four-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$2.29\cdot10^{-2}$	(1,1,1)	21,767	8.20
7	$3.96\cdot10^{-2}$	(1,2,3)	4,888	36.5
9	$4.73\cdot10^{-2}$	(1,2,4)	3,266	54.7
14	$6.36\cdot10^{-2}$	(1,3,5)	1,696	105.2
19	$8.18\cdot10^{-2}$	(1,3,7)	1,137	157.0

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

The approximation to the original control surface gets worse as the generalization factor increases. Two examples for the SOC of 100% and 25% are shown in Figures 6-28 and 6-29, respectively, for L=3. For L=14, the approximations for the same SOC are shown in Figures 6-30 and 6-31. The accuracy decreases as a cell in the associative memory covers more of the input space. The surfaces are not continuous as compared to the two-input controller surfaces, so the generalization factor cannot be increased to the same degree as for the simpler controller. If the function is continuous, a larger generalization factor can be used. If discontinuities exist, a smaller generalization factor must be used to minimize the RMS error. A generalization factor of L=19 did not produce acceptable results. The surface approximation was not accurate enough to produce the desired torque commands. The battery SOC dropped to zero since not enough recharging was generated to complete the mission.

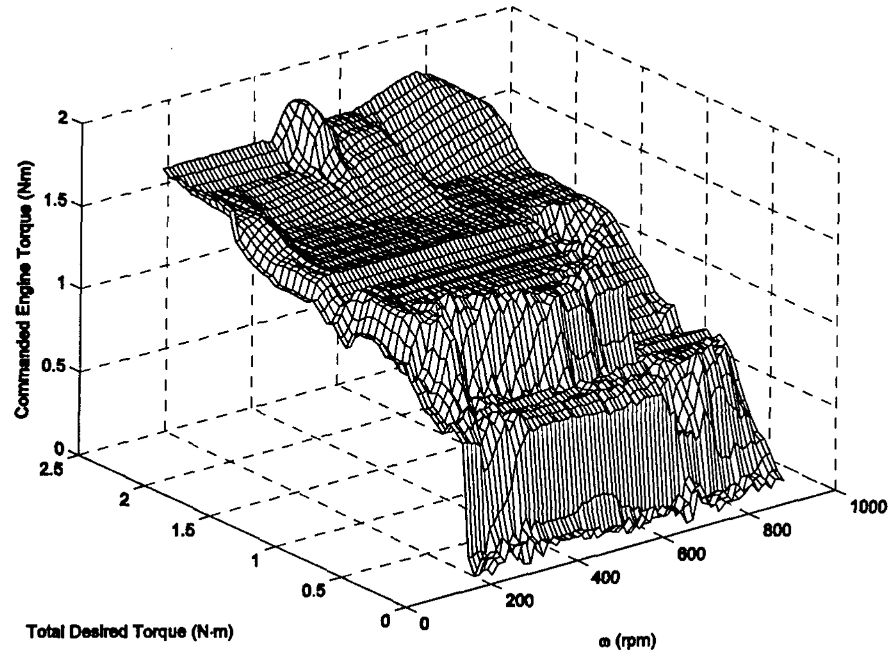


Figure 6-28: CMAC Approximation (L=3) for Engine Torque Control Surface (see Figure 6-12), Four-Stroke, Three-Input Controller, Charge-Sustaining, SOC=100%

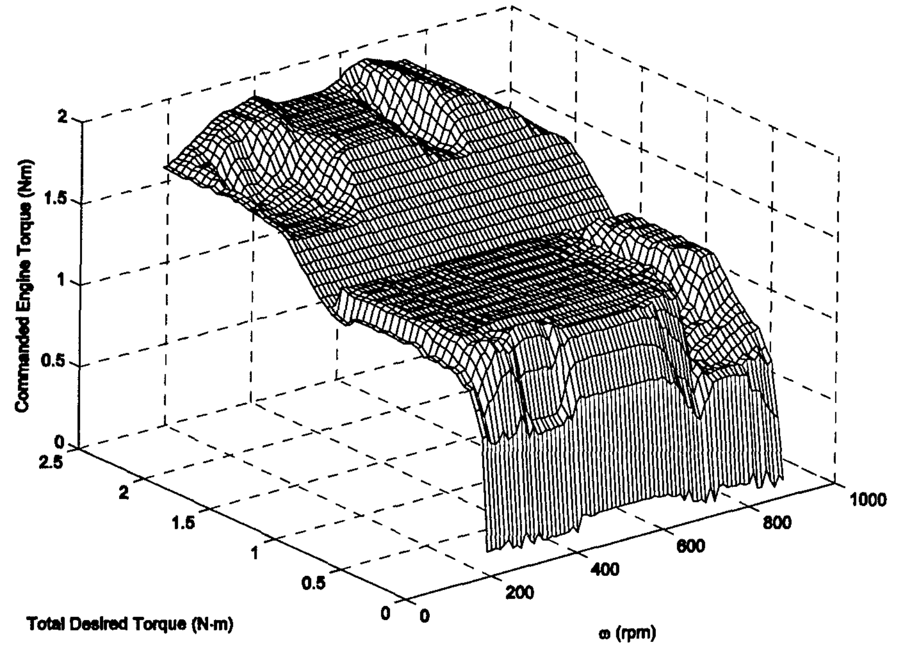


Figure 6-29: CMAC Approximation (L=3) for Engine Torque Control Surface (see Figure 6-13), Four-Stroke, Three-Input Controller, Charge-Sustaining, SOC=25%

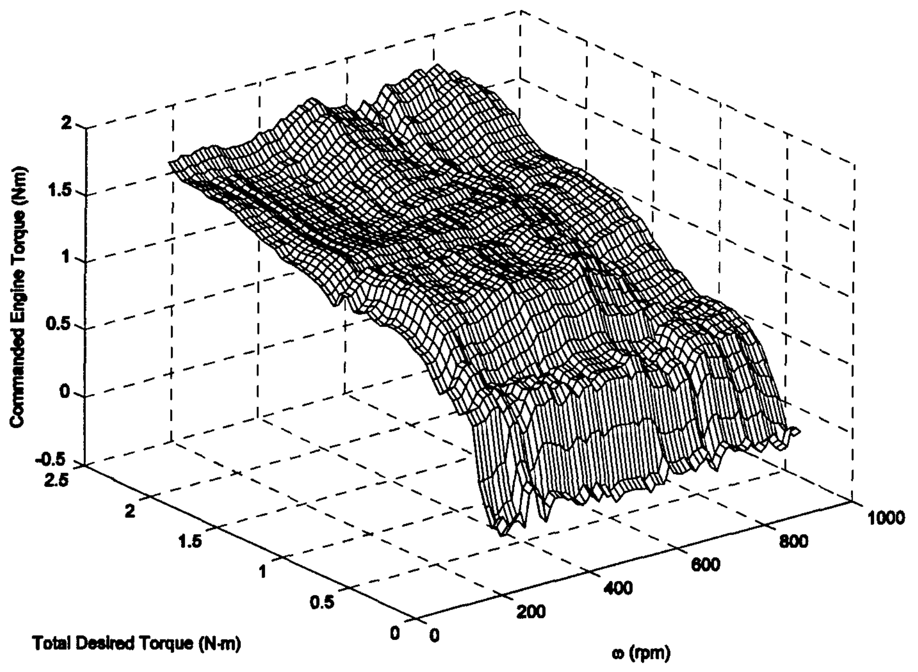


Figure 6-30: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-12), Four-Stroke, Three-Input Controller, Charge-Sustaining, SOC=100%

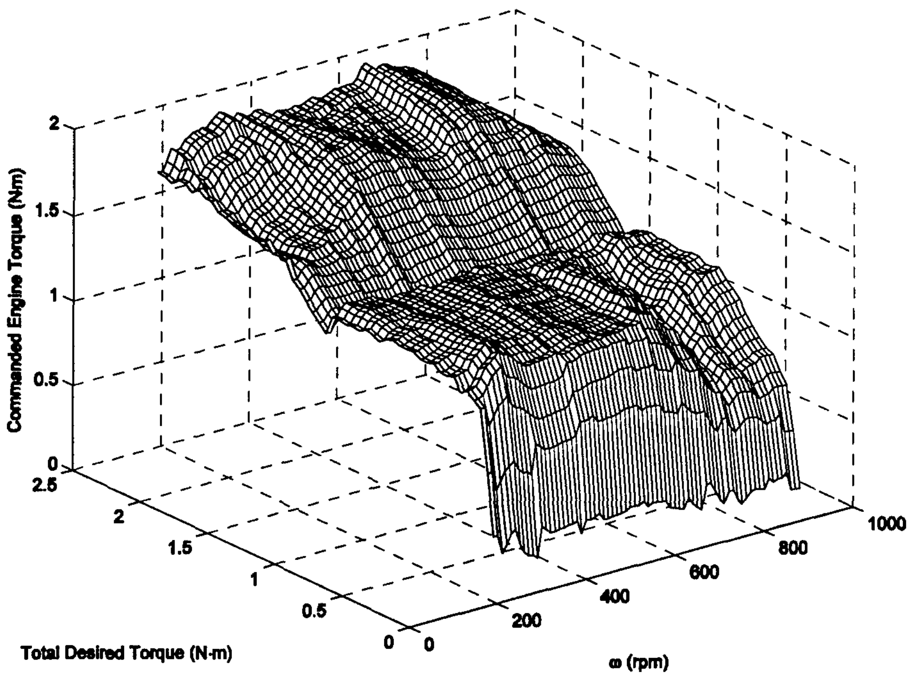


Figure 6-31: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-13), Four-Stroke, Three-Input Controller, Charge-Sustaining, SOC=25%

Two-Stroke Engine, Charge-Depletion Approximations: A summary of the CMAC approximation results for charge-depletion operation using the two-stroke engine is shown in Table 6-6. Alpha and beta values were determined for charge-depletion operation to permit the HEUAV to complete a shorter one-hour mission.

Table 6-6: Summary of the CMAC Approximation Results for the Three-Input Controller, Charge-Depletion, Two-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$2.22 \cdot 10^{-2}$	(1,1,1)	21,767	8.20
7	$3.92 \cdot 10^{-2}$	(1,2,3)	4,888	36.5
9	$4.64 \cdot 10^{-2}$	(1,2,4)	3,266	54.7
14	$6.91 \cdot 10^{-2}$	(1,3,5)	1,696	105.2
19	$8.62 \cdot 10^{-2}$	(1,3,7)	1,137	157.0
25	$1.04 \cdot 10^{-1}$	(1,3,8)	828	215.6

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

The approximation to the original control surface gets worse as the generalization factor increases. Two examples for the SOC of 100% and 25% are shown in Figures 6-32 and 6-33, respectively, for L=14. As a cell in the associative memory covers more of the input space, the approximation gets worse as is revealed by the increase in the RMS error. If discontinuities exist such as in the control surface for the engine, a smaller generalization factor must be used to minimize the RMS error.

The CD surface causes more off-board electrical energy to be used than for the CS surfaces. If the demanded torque is not met by the internal combustion engine, then electrical energy must be used. The CD surfaces do not necessarily keep the engine operating on or near the IOL as often as for the CS surfaces. The weighting factor alpha in the objective function determines the amount of electrical energy that is used.

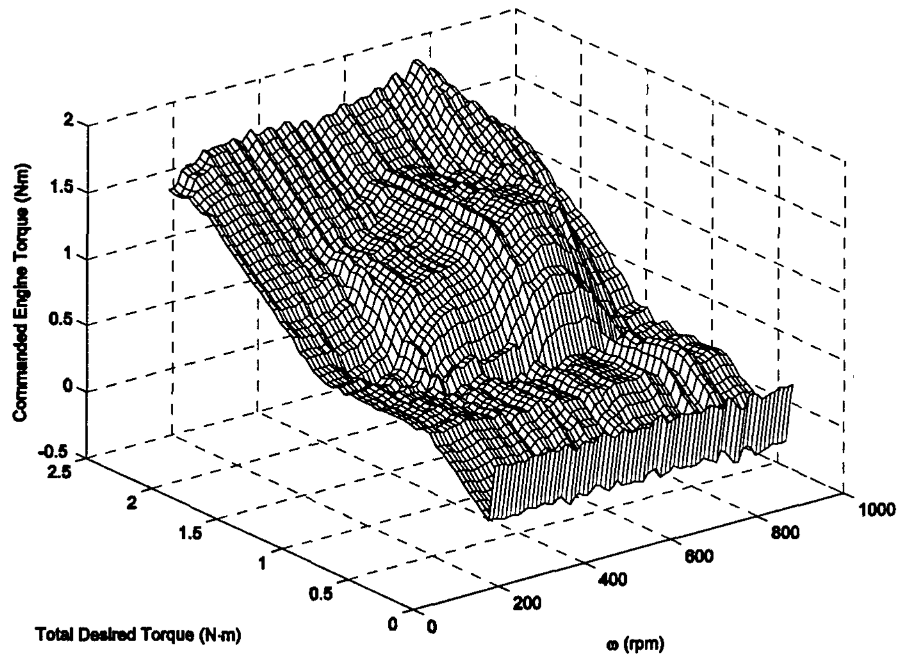


Figure 6-32: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-14), Two-Stroke, Three-Input Controller, Charge-Depletion, SOC=100%

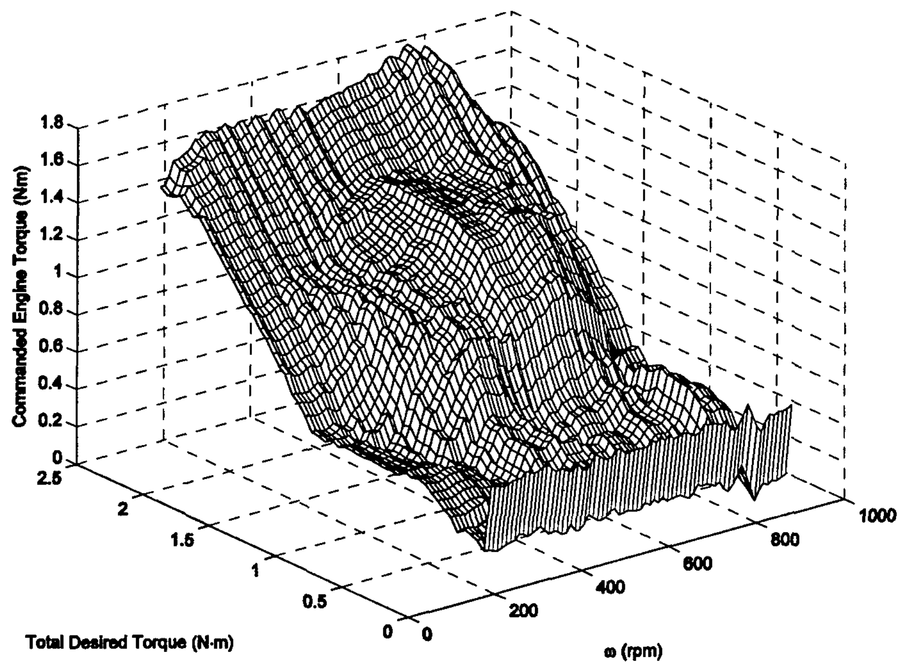


Figure 6-33: CMAC Approximation (L=14) for Engine Torque Control Surface (see Figure 6-15), Two-Stroke, Three-Input Controller, Charge-Depletion, SOC=25%

Four-Stroke Engine, Charge-Depletion Approximations: A summary of the CMAC approximation results for charge-depletion operation using the four-stroke engine is shown in Table 6-7. Alpha and beta values were determined for charge-depletion operation to permit the HEUAV to complete a shorter one-hour mission. The runs for L=3, 7, and 9 used 200 training iterations, and the other runs used 150 iterations.

Table 6-7: Summary of the CMAC Approximation Results for the Three-Input Controller, Charge-Depletion, Four-Stroke Engine

Generalization Parameter	RMS Error	Displacement Vector	Number of Weights	Memory Savings[1]
3	$2.50 \cdot 10^{-2}$	(1,1,1)	21,767	8.20
7	$4.42 \cdot 10^{-2}$	(1,2,3)	4,888	36.5
9	$5.25 \cdot 10^{-2}$	(1,2,4)	3,266	54.7
14	$6.53 \cdot 10^{-2}$	(1,3,5)	1,696	105.2
19	$7.69 \cdot 10^{-2}$	(1,3,7)	1,137	157.0
25	$8.82 \cdot 10^{-2}$	(1,3,8)	828	215.6

[1]Number of entries in a LUT/number of weights for the CMAC associative memory

The approximation to the original control surface gets worse as the generalization factor increases. Two examples for the SOC of 100% and 25% are shown in Figures 6-34 and 6-35, respectively, for L=9. As a cell in the associative memory covers more of the input space, the approximation gets worse as is shown by the increase in the RMS error. If discontinuities exist such as in the control surface for the engine, a smaller generalization factor must be used to minimize the RMS error. The charge-depletion surfaces for L=3, 7, 9, 14, and 19 produced acceptable results. A generalization factor of L=25 allowed the battery SOC to drop to a very low value. Since a cell covers nearly half of the input space for the third input dimension, L=25 is too high of a generalization factor for this application.

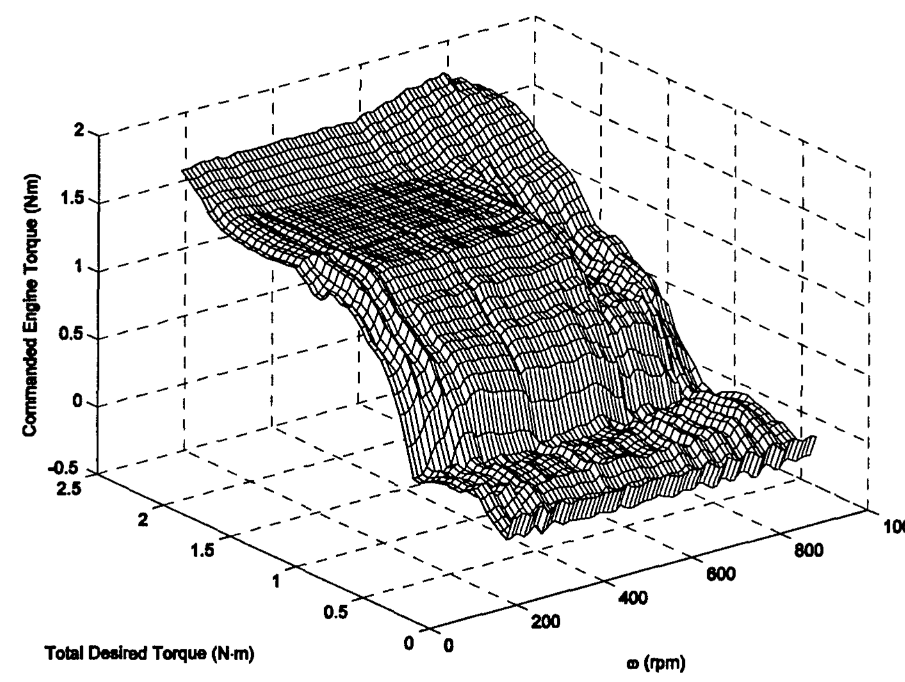


Figure 6-34: CMAC Approximation (L=9) for Engine Torque Control Surface (see Figure 6-16), Four-Stroke, Three-Input Controller, Charge-Depletion, SOC=100%

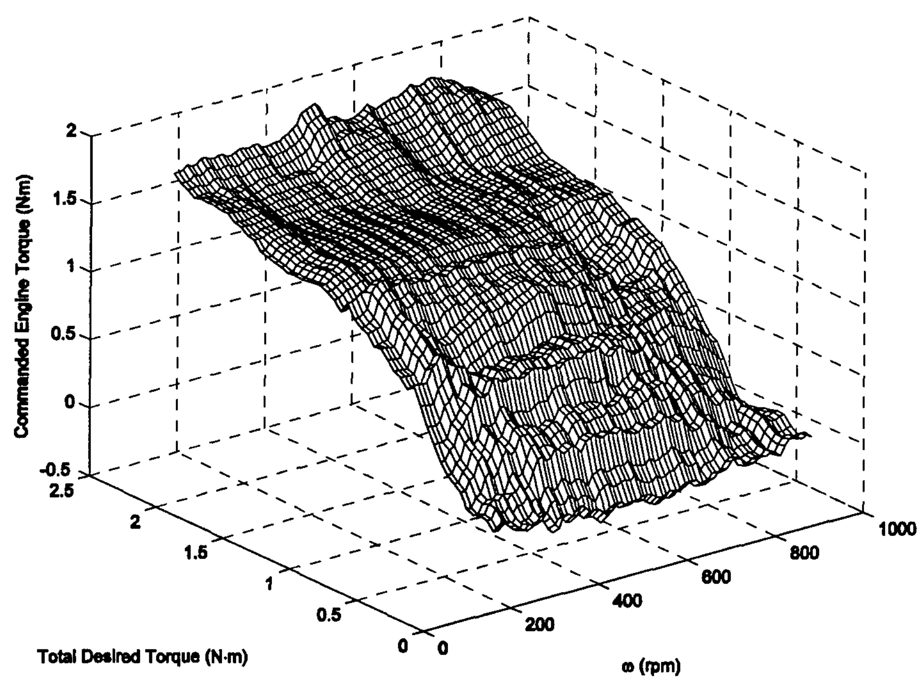


Figure 6-35: CMAC Approximation (L=9) for Engine Torque Control Surface (see Figure 6-17), Four-Stroke, Three-Input Controller, Charge-Depletion, SOC=25%

6.6 Flight Profile Simulation Results

As described in Chapter 5, the small HEUAV is sized for a typical intelligence, surveillance, and reconnaissance (ISR) mission. Two ISR missions and a cruise mission segment with wind turbulence demonstrate the capabilities of the HEUAV.

The MATLAB/Simulink model developed for the HEUAV (see Chapter 4) includes an option for the original UAV configuration that includes an engine only. The engine is sized larger than the engine used in the hybrid-electric version. The results for the HEUAV using the different controllers (rule-based vs. CMAC) will be compared to the UAV with the engine only. Selected plots for each configuration and controller will be shown. The plots include illustrations such as engine torque and speed, motor torque and speed, battery SOC, and UAV speed. The sections will include summary tables of the power consumption and energy use for each configuration and controller for each flight profile.

For the rule-based controller and the two-input CMAC controller, logic must be programmed to determine when charge-sustaining is allowed. The flexibility in the objective function for the three-input controller shows how the CMAC controller can be trained depending on the expected mission length and requirements. Separate logic for the three-input controller to enable charge-sustaining is not needed since the charge sustaining is inherent in the optimization algorithm.

6.6.1: One-Hour ISR Mission

To show initial results for the various mission segments of a flight profile, a short one-hour ISR mission was generated. The speed and altitude for the flight profile is shown in Figure 6-36. The flight profile consists of a take-off, climb, cruise, endurance speed, high speed dash, descent, and landing. The cruise speed is designed to be 50 kts and the endurance speed 25 kts. At endurance speed, the HEUAV operates in electric-only (stealth) mode. The design altitude is approximately 5 kft mean sea level (MSL), and for this flight profile, the UAV takes off from 4 kft MSL. A climb and descent prior to the endurance mission segment simulate the flight over an obstacle.

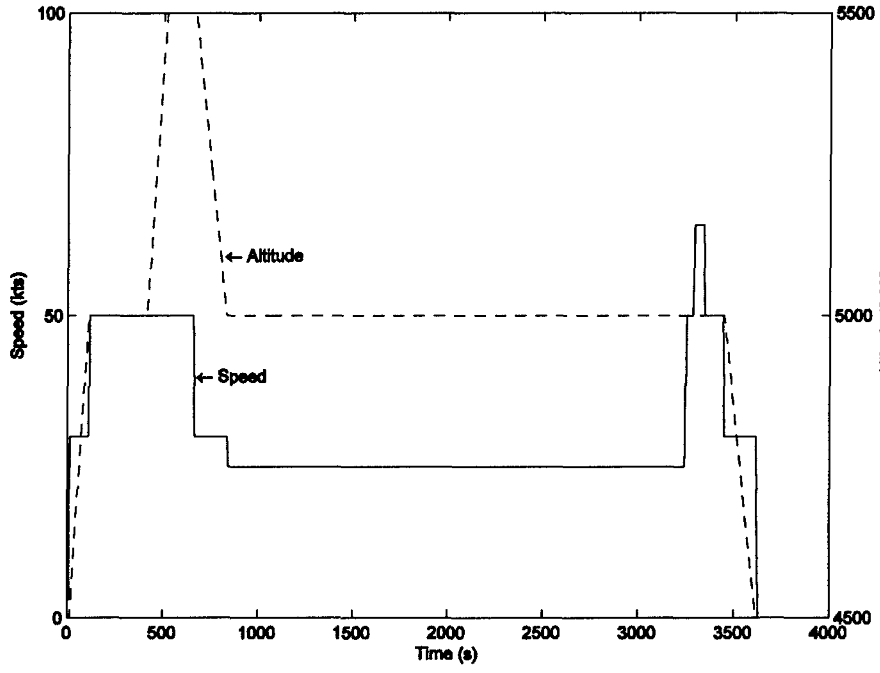


Figure 6-36: One-Hour Flight Profile

Two-Stroke Engine, Charge-Depletion Simulations: A simulation using the original configuration (ICE only) was completed first to provide a baseline for comparison. A plot of the fuel consumed during the ISR mission is shown in Figure 6-37 for the original configuration and the hybrid-electric configuration (rule-based). At the end of the mission, 516 g (18.2 oz) of fuel have been burned for the original configuration. The hybrid-electric configuration has consumed much less fuel. The mass of the UAV is decreased during the simulation as the fuel is burned. The mass of the original configuration will be lighter than the HEUAV at the end of the mission since more fuel is burned. No fuel is burned during the endurance mission segment for the HEUAV since it is in electric-only (stealth) mode. It will be shown that less fuel and less energy will be consumed by the HEUAV.

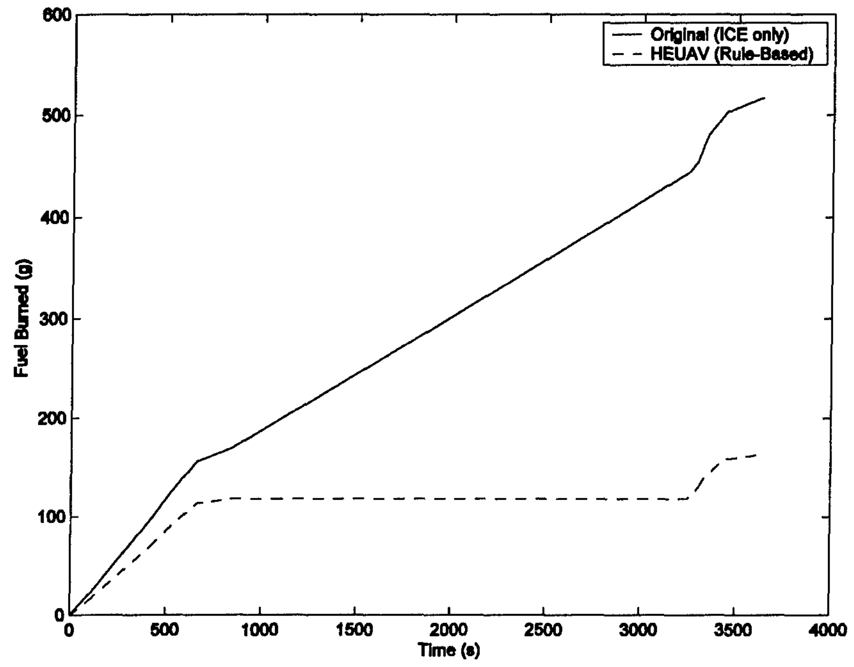


Figure 6-37: Fuel Consumed, One-Hour Mission, Two-Stroke, Original and HEUAV Configurations

The torque and speed of the engine for the HEUAV configuration (rule-based) is shown in Figure 6-38. The speed of the engine slows to zero during electric-only operation. The torque of the engine is also zero during the stealth mode. Since the engine is not operating during approximately half of the mission, the HEUAV configuration intuitively uses much less fuel than the original configuration. Also, no CS is programmed so almost all of the electrical energy in the battery pack is used.

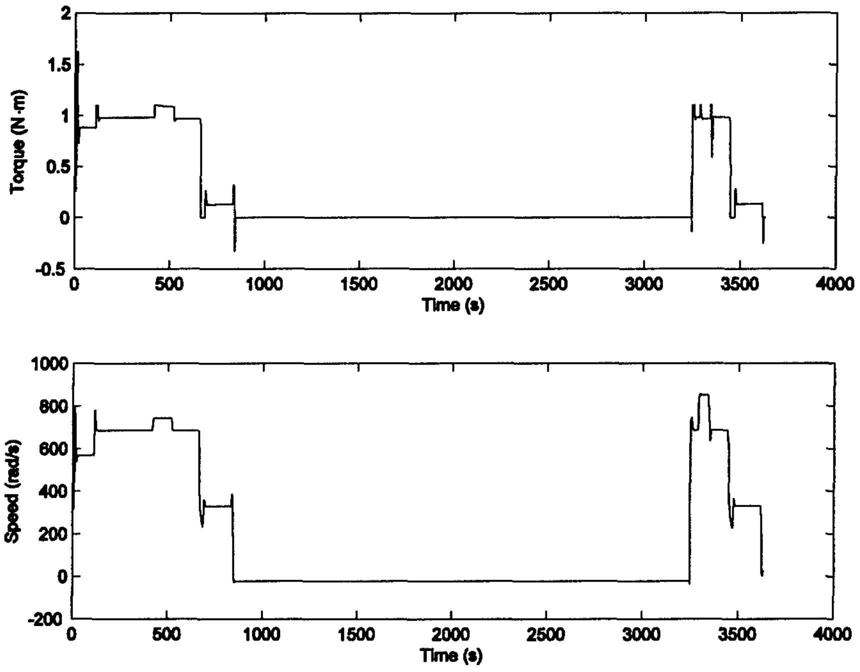


Figure 6-38: Engine Torque and Speed, One-Hour Mission, Two-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Depletion

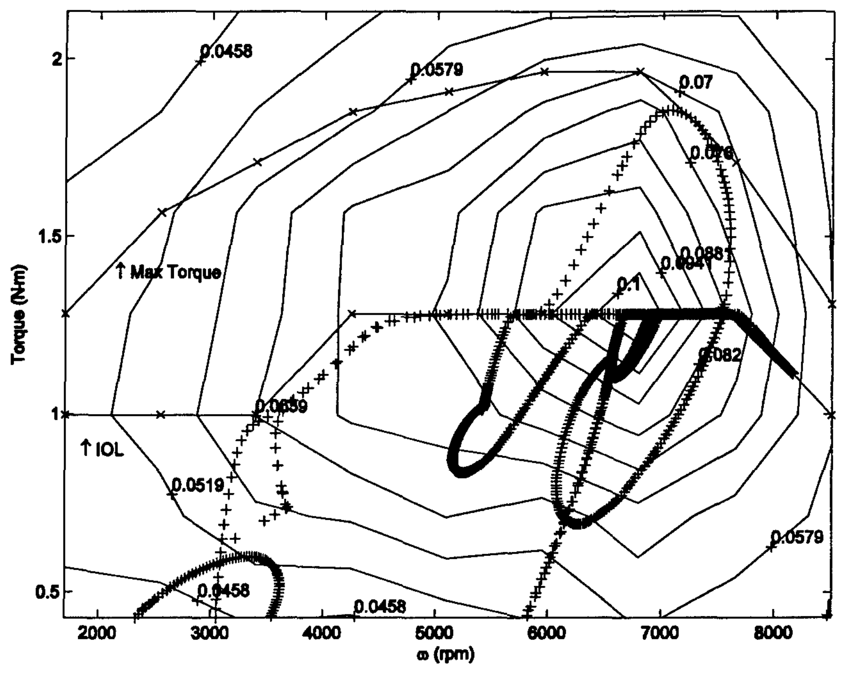


Figure 6-39: Engine Operating Points, One-Hour Mission, Two-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Depletion

The operating points for the ICE for the HEUAV configuration (rule-based) are shown in Figure 6-39. The operating points have been adjusted to sea level. The ICE is operating on the IOL unless the torque demand is less than the IOL torque or if the torque demand is greater than the combined IOL torque and maximum EM torque. The maximum torque from the ICE is used during take-off. Slight variations from the IOL are due to the first-order approximation (function of altitude) for the ICE torque.

The three-input CMAC controller (CD surface) improves on the rule-based controller. The HEUAV using the CMAC controller follows the flight profile closely as it does for the other configurations and controllers. The actual UAV speed vs. the desired speed is shown in Figure 6-40. The pilot/operator model sufficiently keeps the UAV on the flight profile during the ISR mission.

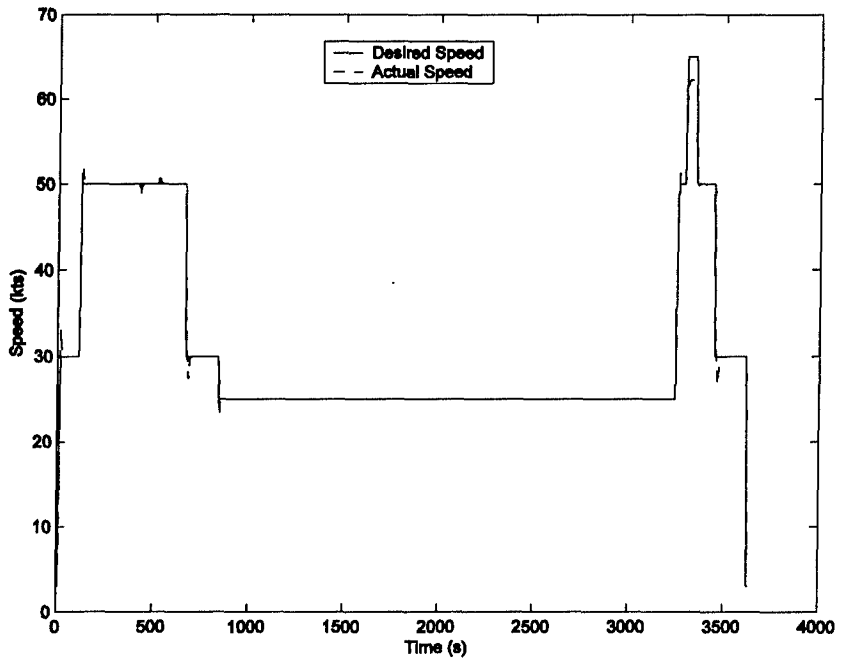


Figure 6-40: Actual vs. Desired UAV Speed, One-Hour Mission, Two-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=19), Charge-Depletion

The intent of the one-hour mission is to efficiently use the electrical energy in the battery pack. For the shorter mission, more electrical energy can be used and the engine does not need to stay on the IOL. Charge-sustaining is not required for this mission. For the three-input CMAC controller, the optimization algorithm produces a surface which allows the engine to operate at a torque output less than the IOL to permit more electrical energy to be used. The engine operating points for the short ISR mission are shown in Figure 6-41.

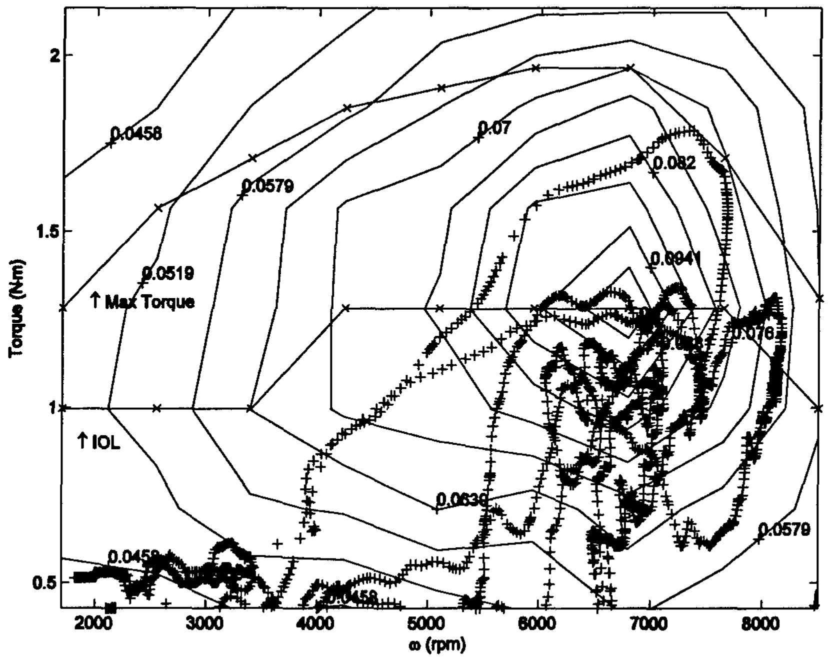


Figure 6-41: Engine Operating Points, One-Hour Mission, Two-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=19), Charge-Depletion

The motor torque and speed are shown in Figure 6-42 for the HEUAV with the three-input CMAC controller (L=19). Recharging is produced during the final descent. During other mission segments, the EM is providing positive torque to the propeller. The torque does drop below zero during some of the transients between the mission segments.

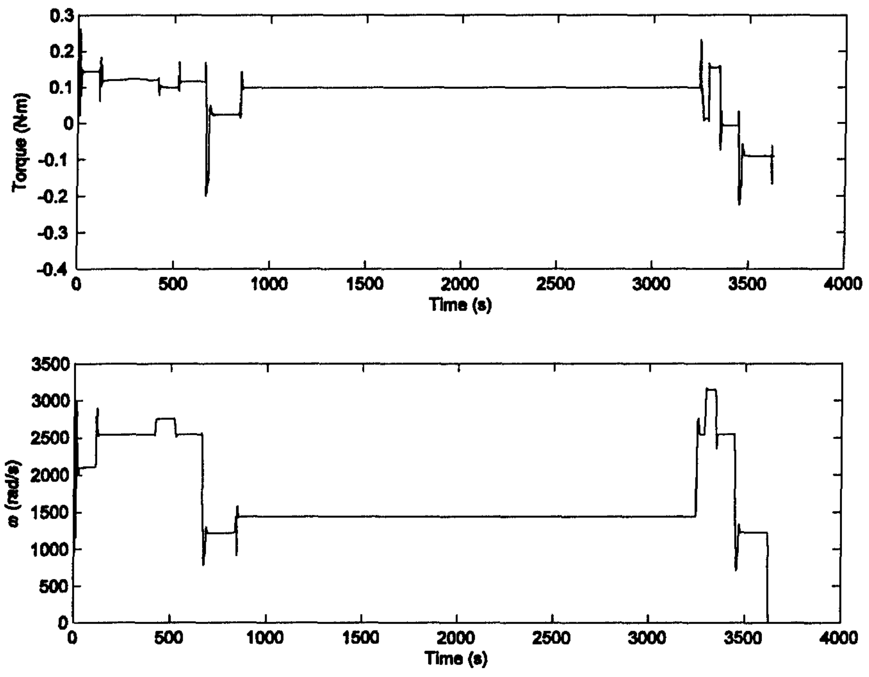


Figure 6-42: Motor Torque and Speed, One-Hour Mission, Two-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=19), Charge-Depletion

The optimization algorithm generates a CD control surface that maximizes the use of the electrical energy. The battery SOC is allowed to drop down to 10-15%. If the SOC is dropped below 10%, little margin is left for emergencies if electrical power is needed for climbing or acceleration. A plot of the SOC for several of the three-input CMAC controllers is shown in Figure 6-43. For L=19, the SOC drops slightly below 10%. The modeling error as L exceeds 20 is too great and too much electrical energy is used. As L increases above 20, the CMAC structure is not well defined, and the modeling error increases greatly. For generalization factors less than 20, the decrease in the SOC is approximately the same.

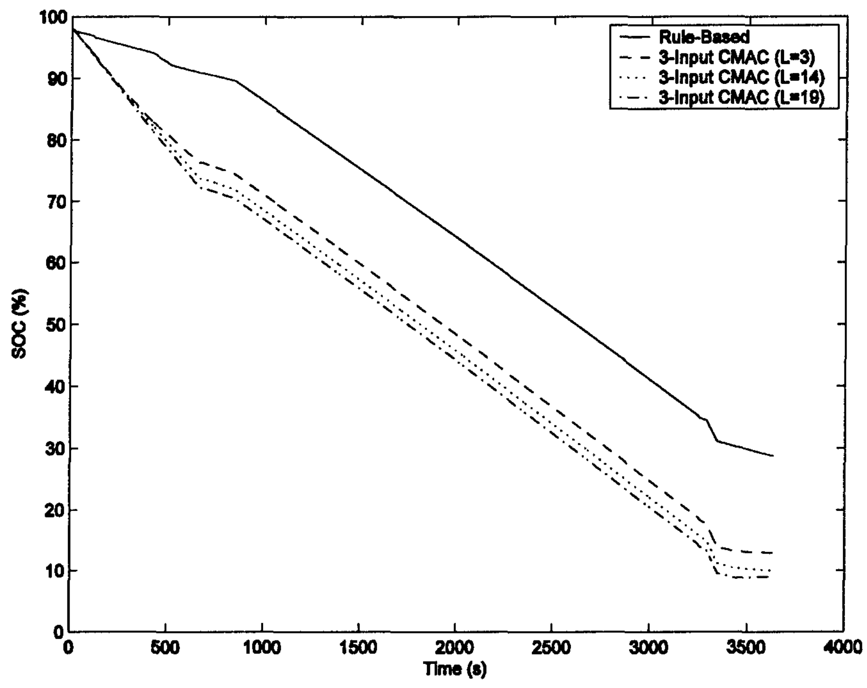


Figure 6-43: Battery SOC, One-Hour Mission, Two-Stroke, HEUAV Configuration, Rule-Based and Three-Input CMAC Controllers, Charge-Depletion

A summary of the power consumption and energy use for the HEUAV during the one-hour ISR mission with the two-stroke engine is given in Table 6-8. The power consumption is broken down by mission segment and provides insight into which mission segment requires the most and the least power. As expected, take-off and the maximum speed dash consume the most power and the descent and endurance speed segment consume the least amount of power. The HEUAV with the three-input CMAC controller (L=19) uses 67% less energy than the original configuration and 6.5% less energy than the rule-based controller. The rule-based controller for the HEUAV uses 65% less energy than the original configuration.

Table 6-8: Power Consumption and Energy Use Summary for One-Hour Flight Profile, Two-Stroke Engine

Mission Segment	Engine Only	Rule-Based	Two-Input CMAC, no Charge-Sustaining			Three-Input CMAC, Charge-Depletion				
			L=3	L=10	L=18	LUT	L=3	L=9	L=14	L=19
Power Consumption										
Take-Off (W)	24274	11460	12118	12099	12118	12145	12202	12143	12195	12071
Climb (W)	8940	6760	6345	6351	6333	4606	4651	4612	4271	4436
Cruise (W)	10255	7448	7257	7257	7256	6857	6732	6475	6567	6315
Endurance (W)	4941	240.3	240.4	240.4	240.4	240.5	240.5	240.5	240.5	240.6
Max Speed (W)	19969	13240	13662	13523	13602	11910	12008	12241	12246	12761
Descent (W)	3259	1224	857	880	872	2684	2714	2725	2525	3180
Energy Use										
Fuel (g)	516.2	162.7	155.4	155.4	155.4	150.5	149.7	147.8	145.7	146.5
Fuel (kWh)	6.14	1.95	1.86	1.86	1.86	1.80	1.79	1.77	1.75	1.76
Electricity (kWh)	NA	0.204	0.227	0.227	0.228	0.244	0.246	0.248	0.253	0.256
Total (kWh)	6.14	2.15	2.09	2.09	2.09	2.05	2.04	2.02	2.00	2.01

Four-Stroke Engine, Charge-Depletion Simulations: A comparison of the desired speed and the actual speed from the simulation for the original four-stroke engine-only configuration is shown in Figure 6-44. The UAV follows the flight profile closely. The engine and torque speed are shown in Figure 6-45. Since the original configuration only has gasoline available, it will use much more energy than the hybrid-electric configuration that has a fully-charged battery pack available.

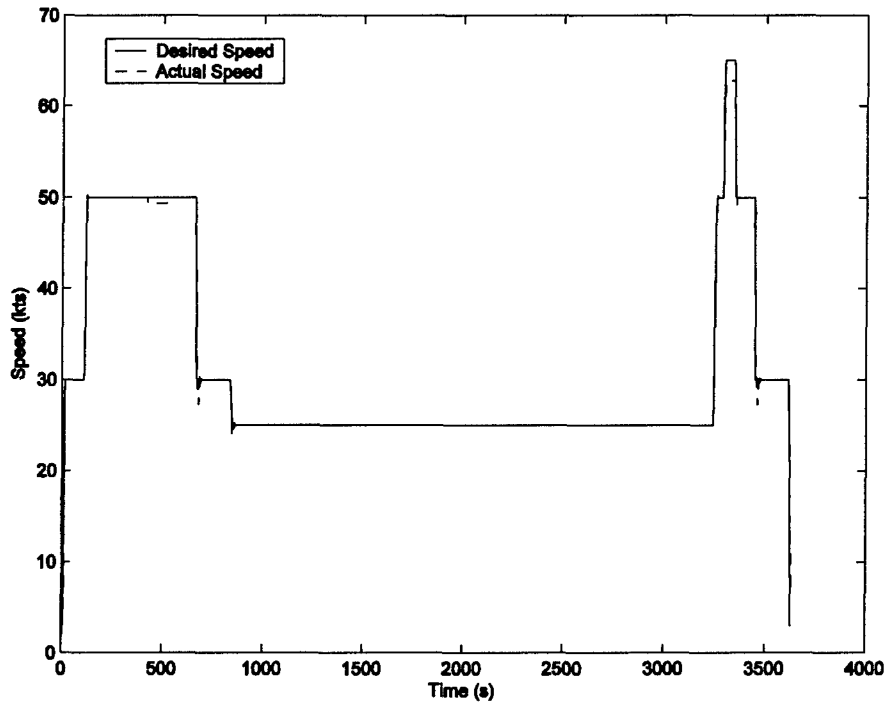


Figure 6-44: Desired and Actual UAV Speed, One-Hour Mission, Four-Stroke, Original Configuration

The hybrid-electric propulsion system controllers use a charge-depletion strategy for the short mission. Due to the short length of the mission, charging is not needed. The rule-based strategy and the two-input CMAC controller do not use any charge-sustaining logic. The three-input controllers use the charge-depletion control surfaces.

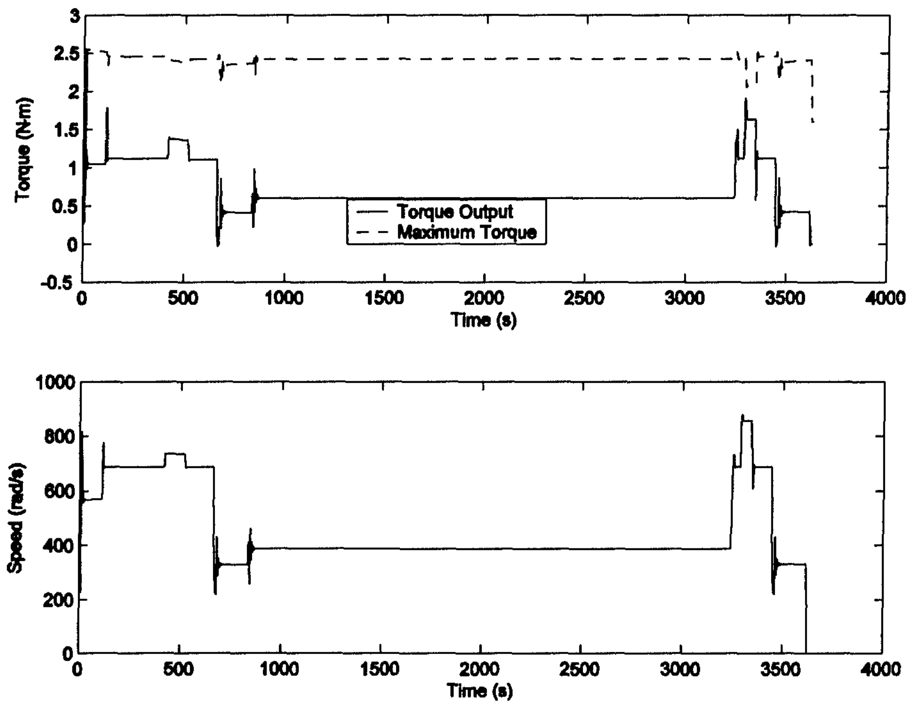


Figure 6-45: Engine Torque and Speed, One-Hour Mission, Four-Stroke, Original Configuration

The rule-based controller is the baseline for the hybrid-electric propulsion system controllers. The EM torque and speed are shown in Figure 6-46. Since a gearbox with a ratio of 3.7 is used, the rotational speed of the motor is greater than the propeller or engine. The two-input CMAC controller approximates the rule-based controller. The three-input controllers improve on the rule-based controller. A sample plot of the battery SOC for each of these controllers is shown in Figure 6-47. All of the CMAC controller surfaces are generated for sea-level operation. First-order approximations are used to adjust the engine torque output commands. The rule-based controller also uses adjusted output torque commands, but due to the variation in implementation and modeling errors, the results will be slightly different than the two-input CMAC controllers.

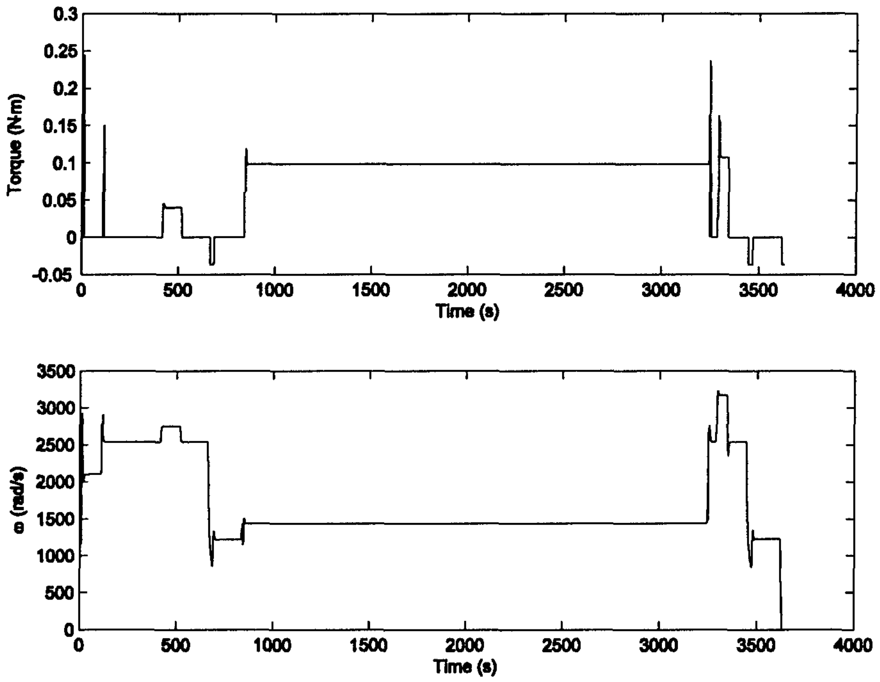


Figure 6-46: Motor Torque and Speed, One-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Depletion

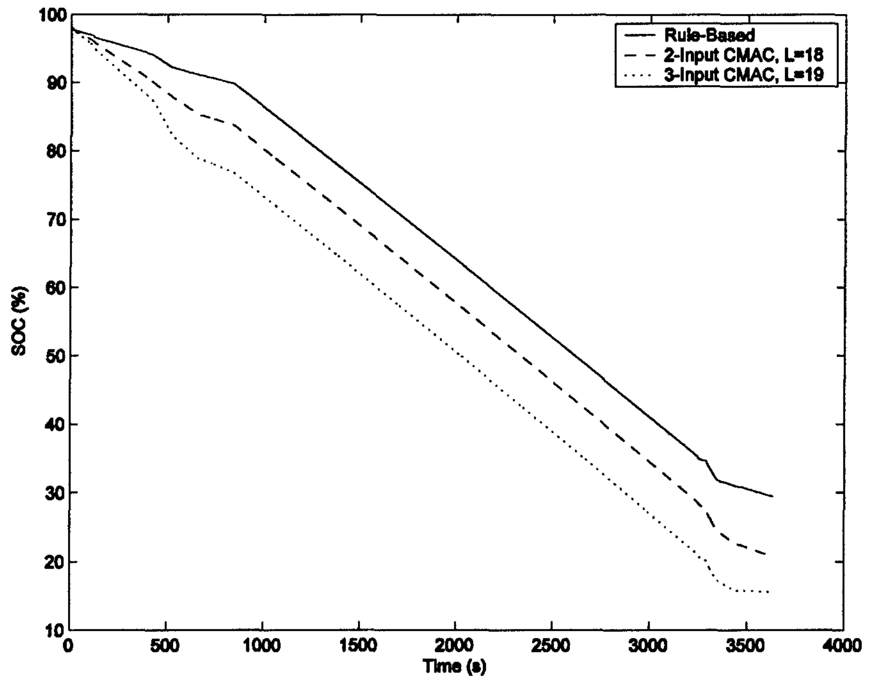


Figure 6-47: Battery SOC, One-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based and CMAC Controllers, Charge-Depletion

The simulation results for the three-input CMAC controllers reveal that they use less electrical energy than the other controllers. For comparison, the engine operating points for the rule-based controller are shown in Figure 6-48. An example of the engine operating points for the three-input CMAC controller (L=19) is shown in Figure 6-49. The torque has been adjusted to sea level. The commands near the IOL do not fall precisely on the IOL due to the CMAC control surface approximation. The rule-based controller surface is continuous, and the change in torque commands is more gradual than for the CMAC controller. The output torque of the engine is often on the IOL during the flight profile for the rule-based controller, but this does not necessarily mean the least amount of energy is being used with the available electrical energy in the battery pack.

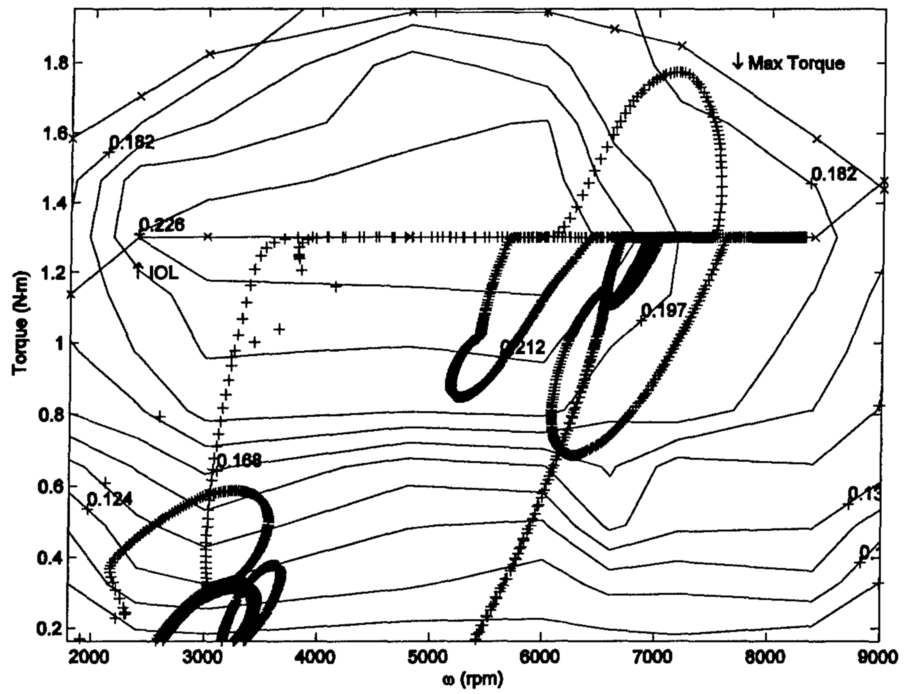


Figure 6-48: Engine Operating Points, One-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Depletion

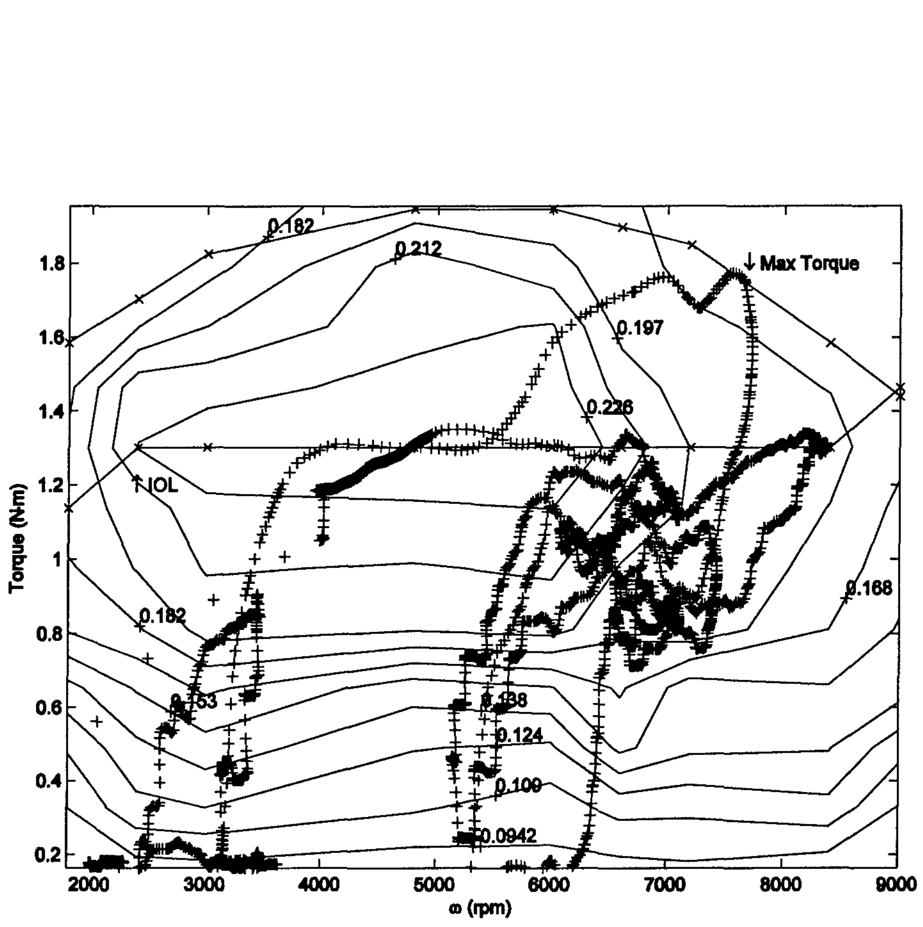


Figure 6-49: Engine Operating Points, One-Hour Mission, Four-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=19), Charge-Depletion

A summary of the power and energy use for the rule-based, two-input, and three-input CMAC controllers is listed in Table 6-9. The control surfaces produced with the higher generalization numbers give satisfactory results (i.e. minimal energy use and larger memory savings), so the surfaces with the lower generalization numbers are not needed. Since the flight profile takes advantage of the charge-depletion logic or controller surfaces, much less energy is used for the hybrid-electric configurations than the original (ICE only) configuration. The rule-based controller uses 54% less energy than the original configuration. The three-input controller (L=19) uses 58% less energy than the original configuration. The large decrease in energy use is due to the charge-depletion surfaces since no gasoline energy is used to maintain the battery charge.

Table 6-9: Power Consumption and Energy Use Summary for One-Hour Flight Profile, Four-Stroke Engine

Mission Segment	Engine Only	Rule-Based	Two-Input CMAC, no Charge-Sustaining					Three-Input CMAC, Charge-Depletion				
			L=3	L=10	L=13	L=15	L=18	LUT	L=3	L=9	L=14	L=19
Power Consumption												
Take-Off (W)	9234	4796	5221	5209	5223	5212	5212	5369	5357	5392	5375	5354
Climb (W)	3020	2425	2245	2248	2247	2247	2245	2279	2202	2355	2374	2362
Cruise (W)	4052	3324	3091	3087	3090	3090	3087	2967	2859	2922	3023	2934
Endurance (W)	1756	241.0	241.1	241.1	241.1	241.1	241.1	241.1	241.2	241.1	241.1	241.2
Max Speed (W)	7464	5549	5667	5658	5647	5648	5647	5629	5632	5642	5548	5738
Descent (W)	1151	588.0	456.2	453.9	448.2	459.1	455.4	819.8	979.7	920.6	919.1	944.6
Energy Use												
Fuel (g)	192.3	70.4	64.1	63.9	63.9	64.0	63.9	61.2	60.5	61.0	61.7	60.3
Fuel (kWh)	2.29	0.844	0.768	0.766	0.766	0.767	0.766	0.734	0.725	0.732	0.740	0.723
Electricity (kWh)	N/A	0.202	0.226	0.226	0.226	0.226	0.226	0.235	0.238	0.238	0.236	0.239
Total (kWh)	2.29	1.05	0.994	0.993	0.993	0.993	0.993	0.968	0.963	0.969	0.976	0.962

6.6.2: Three-Hour ISR Mission

A more realistic and appropriate three-hour ISR mission was generated to test the various controllers. The flight profile is shown in Figure 6-50. The flight profile includes a take-off, climb, cruise, endurance speed, high speed dash, descent, and landing. The ISR segment is split into two different segments to simulate the observation of two different ground locations. The cruise speed is designed to be 50 kts and the endurance speed 25 kts. At endurance speed, the hybrid-electric UAV operates in electric-only (stealth) mode. The design altitude is approximately 5 kft mean sea level (MSL), and for this flight profile, the UAV takes off from 3.5 kft MSL. Simulations will be given for the two-stroke and four-stroke engine designs.

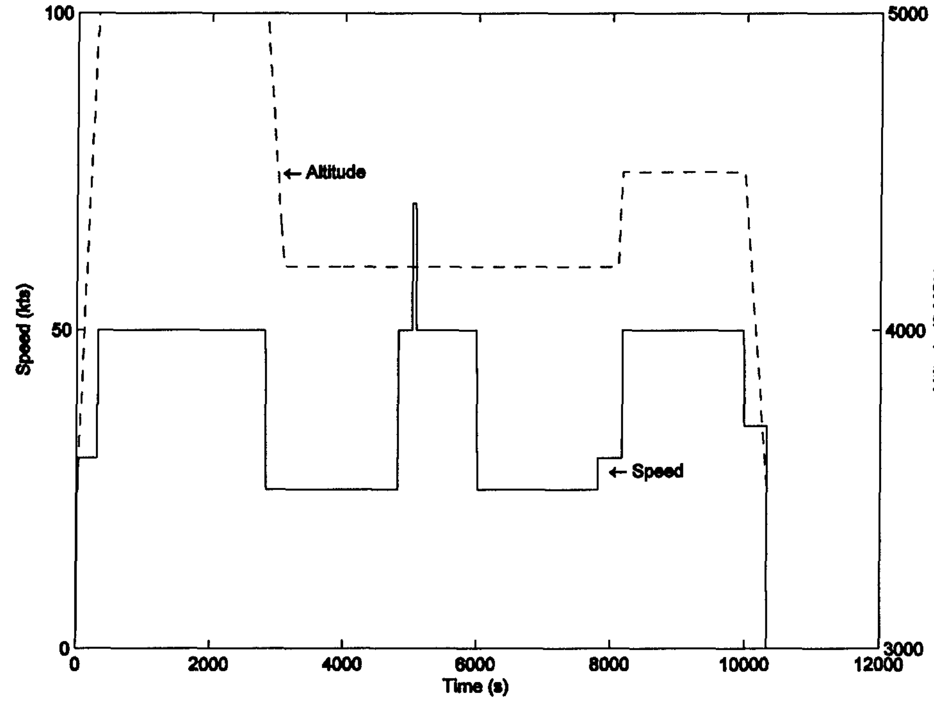


Figure 6-50: Three-Hour Flight Profile

Two-Stroke Engine, Charge-Sustaining Simulations: Simulations were completed for the three-hour mission using the two-stroke engine and the various configurations and controllers.

The torque output and the maximum torque of the ICE for the original configuration are shown in Figure 6-51. The torque is adjusted for the altitude and for an altitude of 5 kft MSL, the output torque of the engine is approximately 86% of the output torque at sea level using a first-order approximation. During the maximum speed dash, the engine operates at its maximum torque output. The engine speed during the ISR mission is shown in Figure 6-52. The rotational speed nears 900 rad/s during the high speed dash. During the rest of the mission, the speed stays above 200 rad/s except during the landing and shutdown.

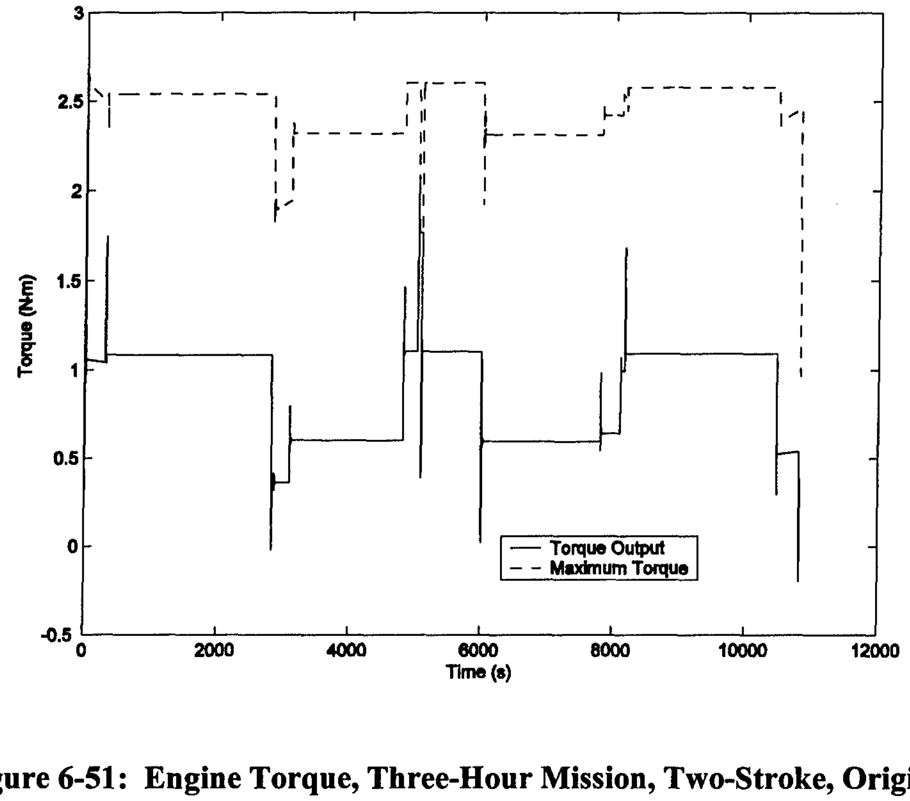


Figure 6-51: Engine Torque, Three-Hour Mission, Two-Stroke, Original Configuration

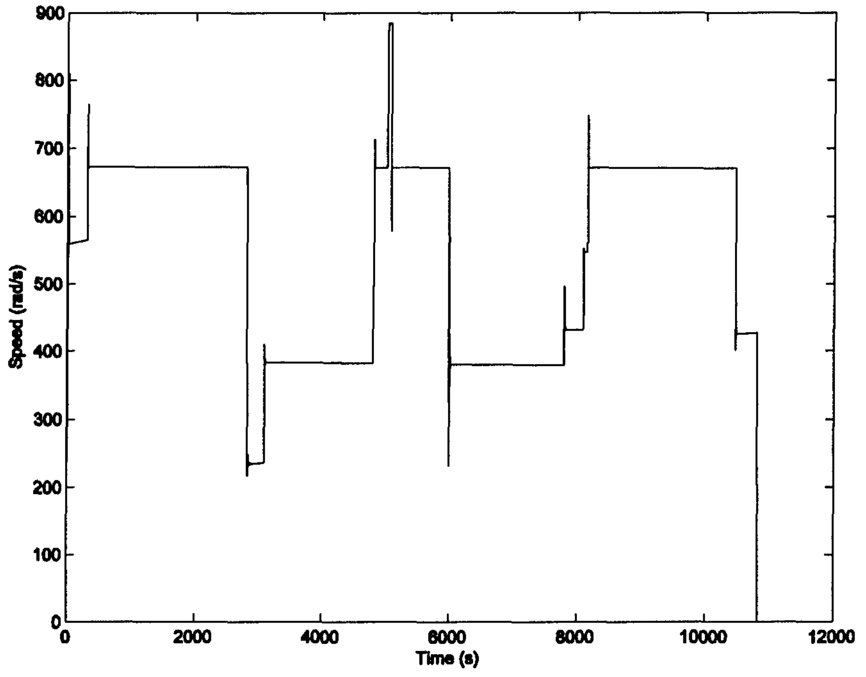


Figure 6-52: Engine Speed, Three-Hour Mission, Two-Stroke, Original Configuration

The rule-based controller was used in the simulations to provide a baseline for the HEUAV results. Charge-sustaining is allowed through-out the mission to provide enough electrical energy during the two half-hour ISR mission segments. The amount of charge-sustaining is dependent on the battery SOC since a proportional-derivative (PD) controller controls the amount of extra engine torque. Battery plots, including the SOC, voltage, and current are shown in Figure 6-53. Due to the PD controller, the rate of charging increases as the SOC decreases. The SOC does not drop below 15% due to the programmed logic. The rule-based controller manages the storage of electrical energy sufficiently for the hybrid-electric system, but the three-input CMAC neural network controller improves on the rule-based controller.

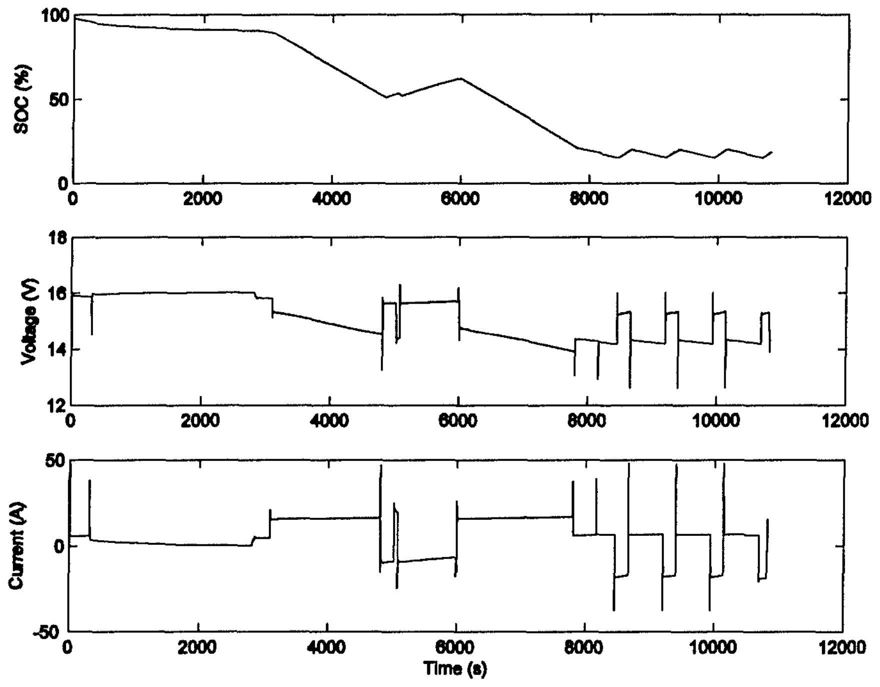


Figure 6-53: Battery Plots, Three-Hour Mission, Two-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

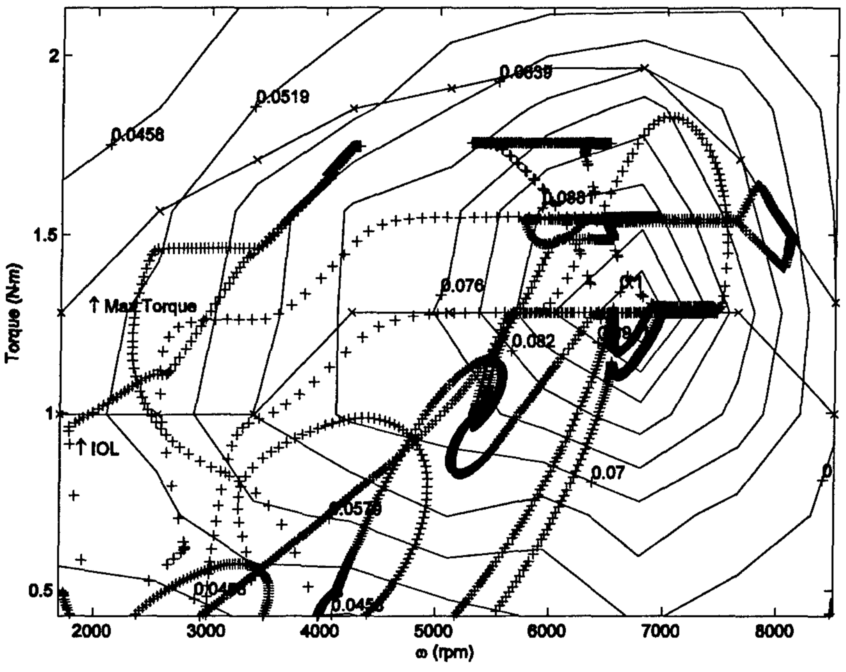


Figure 6-54: Engine Operating Points, Three-Hour Mission, Two-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

The rule-based controller operates the engine on the IOL during moderate torque demands and less than the IOL during low torque demands. In Figure 6-54, the operating points for the engine are shown on the efficiency map for the two-stroke engine. The operating points are adjusted to sea level. The engine operates near the maximum torque during the high-speed dash and acceleration. During the recharging at the end of the mission, the engine torque output is increased above the IOL to a lower efficiency. This relatively large increase in torque due to the PD controller will be shown using the CMAC controller not to be the best method for recharging at the end of the mission. The rule-based controller provides a good baseline for comparison to the three-input CMAC controller results.

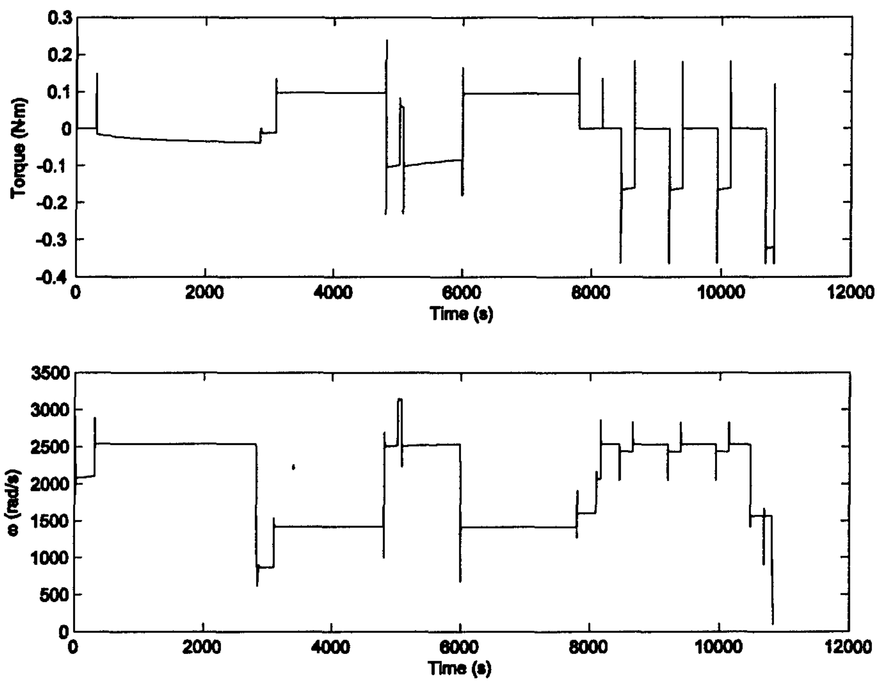


Figure 6-55: Motor Torque and Speed, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

The motor torque and speed are shown in Figure 6-55. The gear unit has a ratio of 3.7 so the motor operates at a much higher speed than the engine and the propeller. Negative torque represents recharging as seen during the descent and charge-sustaining operation. During the descents, the amount of recharging can be varied. The plots illustrate that the motor is sufficiently sized for the HEUAV application.

Simulations using the optimization results and the three-input CMAC approximation controllers were completed for L=3, 7, 9, 14, and 19. For L=19, the modeling error was relatively large and too much recharging was created so the battery SOC went to 100%. The additional torque from the engine was then "wasted" and not used to recharge the battery pack. The UAV follows the speed and altitude profile closely for the three-input CMAC controller (L=14) as shown in Figure 6-56. The battery SOC for the different controllers is shown in Figure 6-57. During the constant speed and altitude mission segments (i.e. cruise), an error in the approximation will cause a difference in the amount of recharging. For L=3, 7, 9 and 14, the battery SOC follows a similar path as for the optimization (i.e. LUT) results. All of the simulations end with a battery SOC near 20% so all of the approximations are sufficient.

A closer look at Figure 6-57 reveals that the recharging is completed at a relatively constant rate as compared to the rule-based controller. Since the rule-based controller uses a PD algorithm, the rate increases as the SOC decreases. For the CMAC controllers, the rate is relatively constant throughout the SOC range. This keeps the engine running at a more continuous power level. Since the CMAC controllers keep the SOC near a constant SOC, most of the recharging is used to provide power to the avionics, flight control system, and the payload.

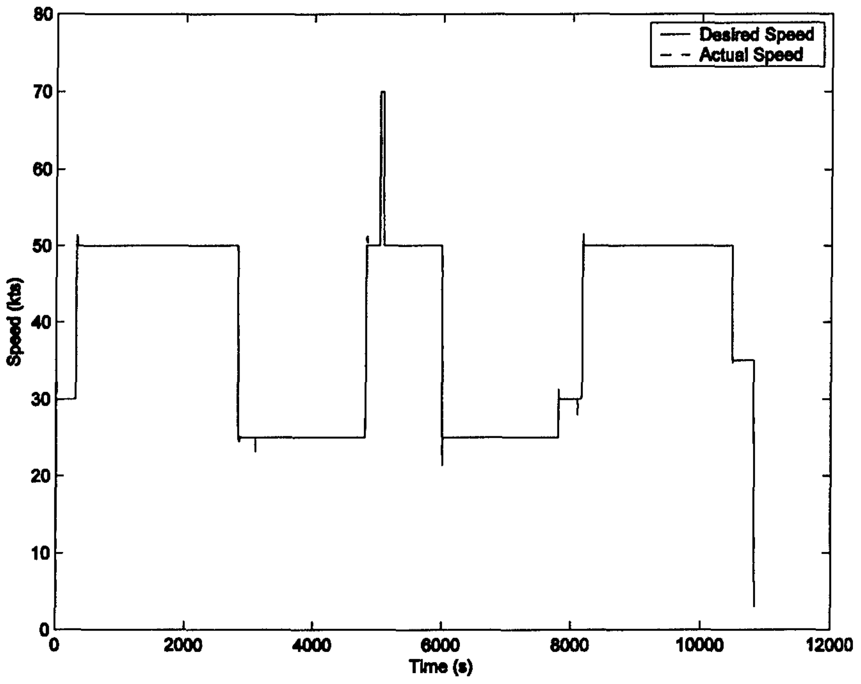


Figure 6-56: Desired and Actual UAV Speed, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=14), Charge-Sustaining

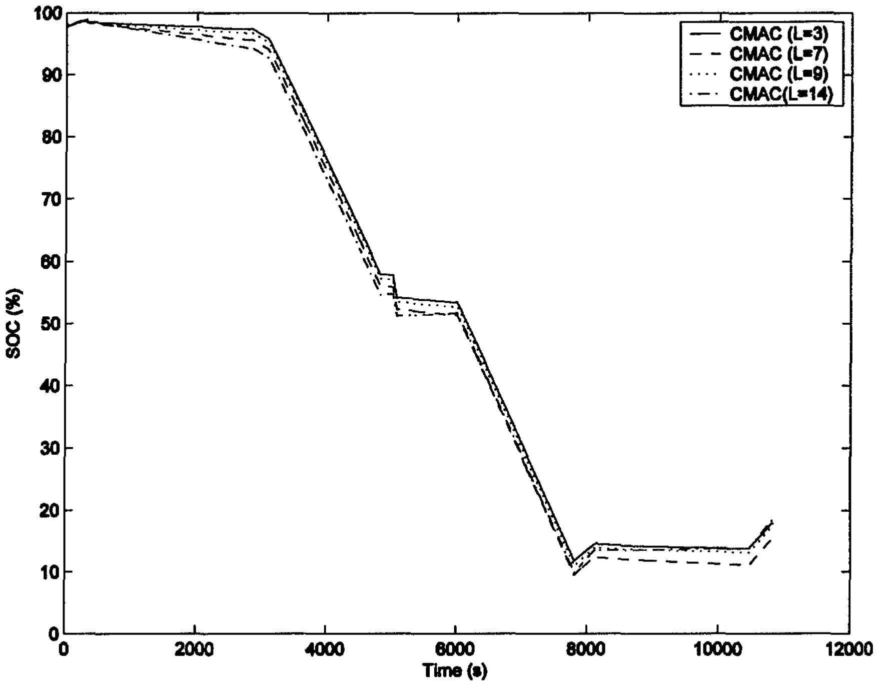


Figure 6-57: Battery SOC, Three-Hour Mission, Two-Stroke, HEUAV Configuration, Three-Input CMAC Controllers, Charge-Sustaining

The power and energy consumption for the HEUAV using the different controllers is shown in Table 6-10. The power consumption is listed for each mission segment. The fuel use is converted to an electricity equivalent. The energy use shows that the original configuration uses the most energy. The energy use for the HEUAV using the rule-based controller is 34% less than the original configuration. The optimization algorithm result (i.e. LUT) uses 38% less energy than the original configuration and 6.9% less than the HEUAV with the rule-based controller. The power consumption values for cruise are taken from the mid-section of the flight profile. For the rule-based controller, the amount of recharging is much greater at the end of the mission than at the beginning of the mission. The CMAC approximations are close to the optimization results and show that a generalization factor up to 14 can be used. Values of L=3, 7, 9, or 14 still provide reasonable results and use less energy as compared to the rule-based controller. In summary, a CMAC controller (L=14) would save two orders of magnitude on memory and would use 6.3% less energy than the rule-based controller.

Table 6-10: Power Consumption and Energy Use Summary for Three-Hour Flight Profile, Two-Stroke Engine

Mission Segment	**Engine Only**	**Rule-Based**	**Three-Input CMAC, Charge-Sustaining**				
			LUT	L=3	L=7	L=9	L=14
Power Consumption							
Take-Off (W)	24935	11412	12512	12505	12531	12628	12573
Climb (W)	8979	6802	7853	7834	7759	7751	7934
Cruise (W)	10479	9790	7640	7642	7643	7641	7848
Endurance (W)	4883	237.0	236.9	236.9	237.0	236.9	237.0
Max Speed (W)	26527	17616	18417	18410	18405	18407	18401
Descent (W)	5054	5534	5338	5355	5467	5457	5644
Energy Use							
Fuel (g)	2025	1318	1220	1224	1216	1223	1234
Fuel (kWh)	24.1	15.8	14.6	14.7	14.6	14.7	14.8
Electricity (kWh)	NA	0.241	0.252	0.250	0.257	0.253	0.256
Total (kWh)	24.1	16.0	14.9	14.9	14.8	14.9	15.0

Four-Stroke Engine, Charge-Sustaining Simulations: A comparison of the desired speed and the actual speed from the simulation for the original four-stroke engine-only configuration is shown in Figure 6-58. The UAV follows the flight profile closely with a maximum speed of 68 kts. The maximum speed is not necessarily limited by the torque of the engine, but it is also dependent on the choice of the propeller and other components for the UAV. The flight profile illustrates that the pilot/operator model is sufficient to keep the UAV on the speed and altitude trace.

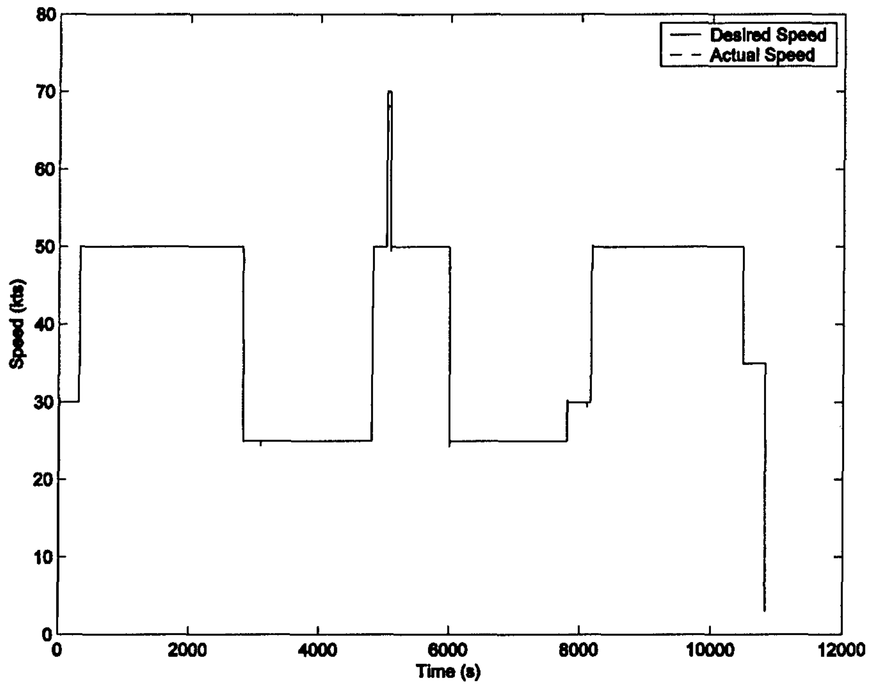


Figure 6-58: Desired and Actual UAV Speed, Three-Hour Mission, Four-Stroke, Original Configuration

Simulations using the original configuration are useful for designing the input space required for the CMAC controllers. By determining the range of various parameters, an efficient structure can be developed for the CMAC neural network. The torque output and the maximum torque of the engine for the original configuration are shown in Figure 6-59. The torque is adjusted for the altitude and for an altitude of 5 kft MSL, the output torque of the engine is approximately 86% of the output torque at sea level using a first-order approximation. During cruise, the engine operates close to its IOL torque. However, during the endurance segment, the engine is forced to operate in a lower efficiency region. During the maximum speed dash, the engine operates at its maximum torque output. The propeller speed (engine speed) is shown in Figure 6-60. The propeller speed varies from 203-930 rad/s. The range of the propeller speed and the range of the demanded torque are useful for determining the range of inputs required for the CMAC controller input space.

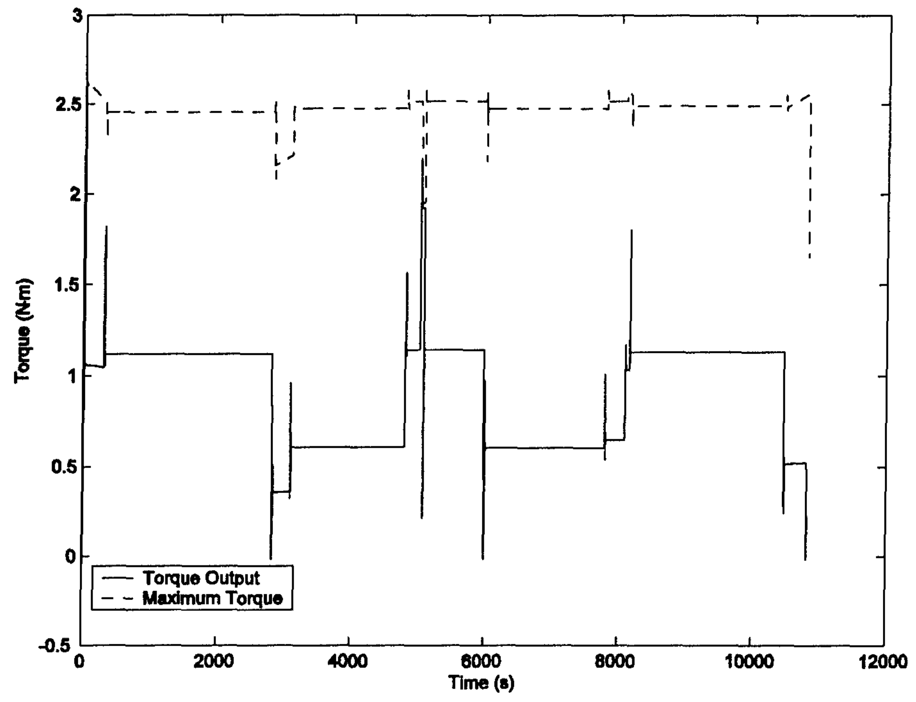


Figure 6-59: Engine Torque, Three-Hour Mission, Four-Stroke, Original Configuration

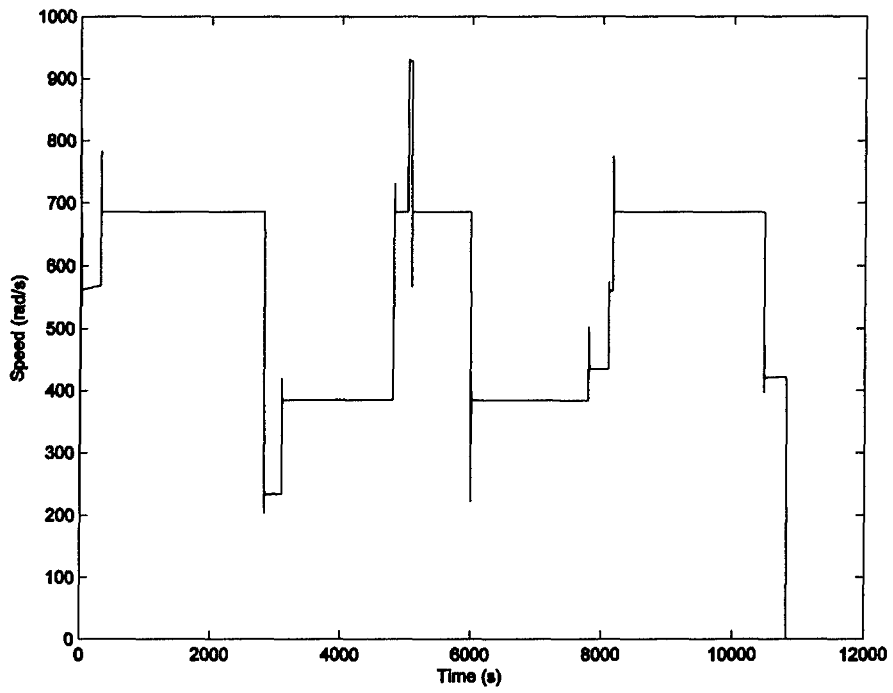


Figure 6-60: Propeller Speed, Three-Hour Mission, Four-Stroke, Original Configuration

The rule-based controller was used in the simulations to provide a baseline for the HEUAV results. Charge-sustaining is allowed through-out the mission to provide enough electrical energy during the two half-hour ISR mission segments. The amount of charge-sustaining is dependent on the battery SOC since a proportional-derivative (PD) controller controls the amount of extra torque the engine produces for recharging. Battery plots, including the SOC, voltage, and current are shown in Figure 6-61. Due to the PD controller, the rate of charging increases as the SOC decreases. The SOC does not drop below 15% due to the programmed logic. The rule-based controller manages the storage of electrical energy sufficiently for the hybrid-electric system, but the CMAC neural network controller improves on the rule-based controller.

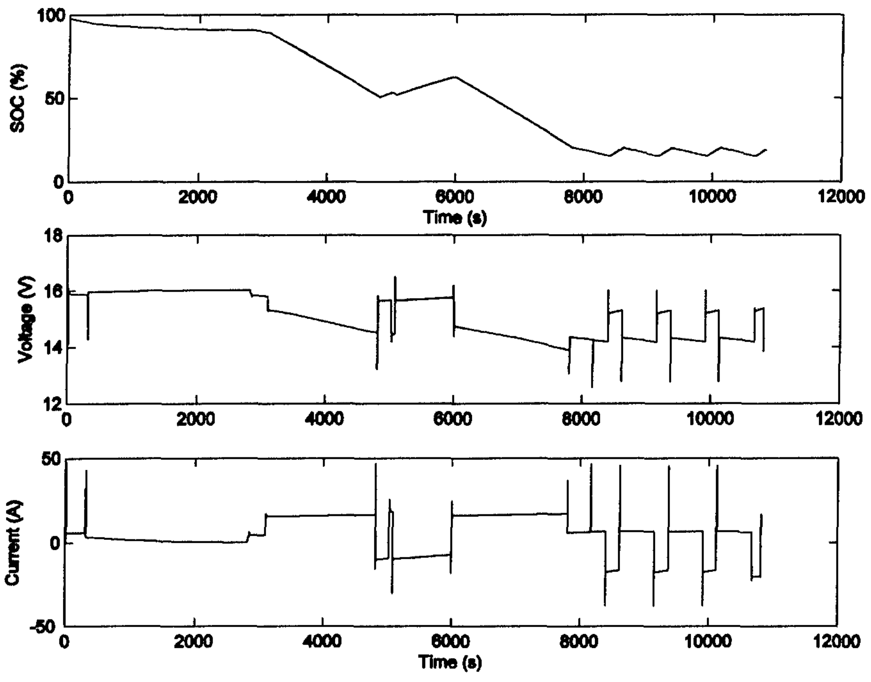


Figure 6-61: Battery Plots, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

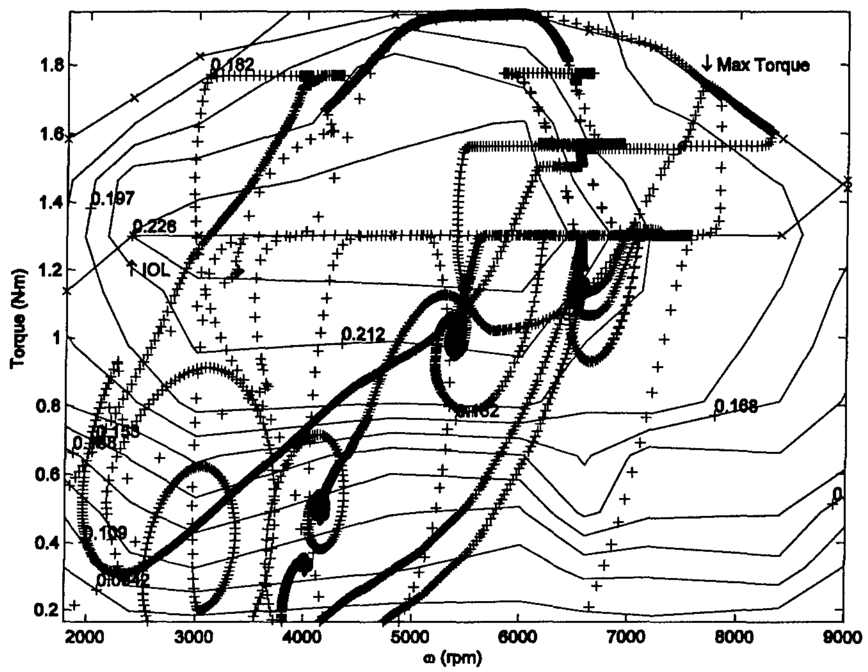


Figure 6-62: Engine Operating Points, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

The rule-based controller operates the engine on the IOL during moderate torque demands and less than the IOL during low torque demands. In Figure 6-62, the operating points for the engine are shown on the efficiency map for the four-stroke engine. The IOL torque and the maximum torque are adjusted for sea level. Since there is variation in the altitude during the flight profile, the IOL and maximum torque vary based on the first-order approximation. The engine operates near the maximum torque during the high-speed dash and acceleration. The rule-based controller provides a good baseline for comparison for the three-input CMAC controller results.

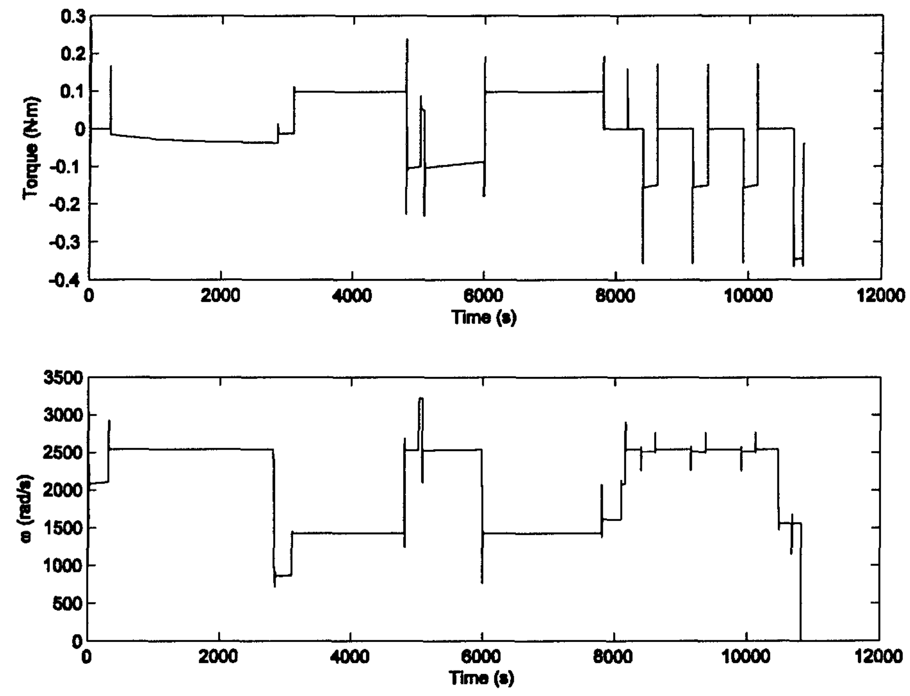


Figure 6-63: Motor Torque and Speed, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Rule-Based Controller, Charge-Sustaining

The motor torque and speed are shown in Figure 6-63. The gear unit has a ratio of 3.7, so the motor operates at a much higher speed than the engine and the propeller. Negative torque represents recharging as seen during the descent and during charge-sustaining operation. During the descents, the amount of recharging can be varied due to a saturation limit on the regeneration command. The plots illustrate that the motor is sufficiently sized for the HEUAV application.

Simulations using the optimization results and the three-input CMAC approximation controllers were completed for L=3, 7, 9, 14, and 19. For L=19, the torque commands were not sufficient for the HEUAV to complete the mission. The UAV follows the speed and altitude profile well for the three-input CMAC controller, L=14, as shown in Figure 6-64. The battery SOC for the different controllers is shown in Figure 6-64. During the constant speed and altitude mission segments (i.e. cruise), an error in the approximation will cause a difference in the amount of recharging. For L=19, the approximation causes insufficient recharging. However, for L=3, 7, 9 and 14, the battery SOC follows a similar path as for the optimization (i.e. look-up table) results. The battery SOC for the CMAC controller (L=9, 14) drops below 10% due to not enough recharging during cruise. All of the simulations end with a battery SOC near 20%. This permits a relatively accurate comparison to be made between the different controllers.

A closer look at Figure 6-64 reveals that recharging is completed at a relatively constant rate as compared to the rule-based controller. Since the rule-based controller uses a PD algorithm, the rate increases as the SOC decreases. For the CMAC controllers, the rate is relatively constant throughout the SOC range. This keeps the engine running at a more continuous power level. Since the CMAC controllers keep the SOC near a constant SOC, most of the recharging is used to provide power to the avionics, flight control system, and the payload.

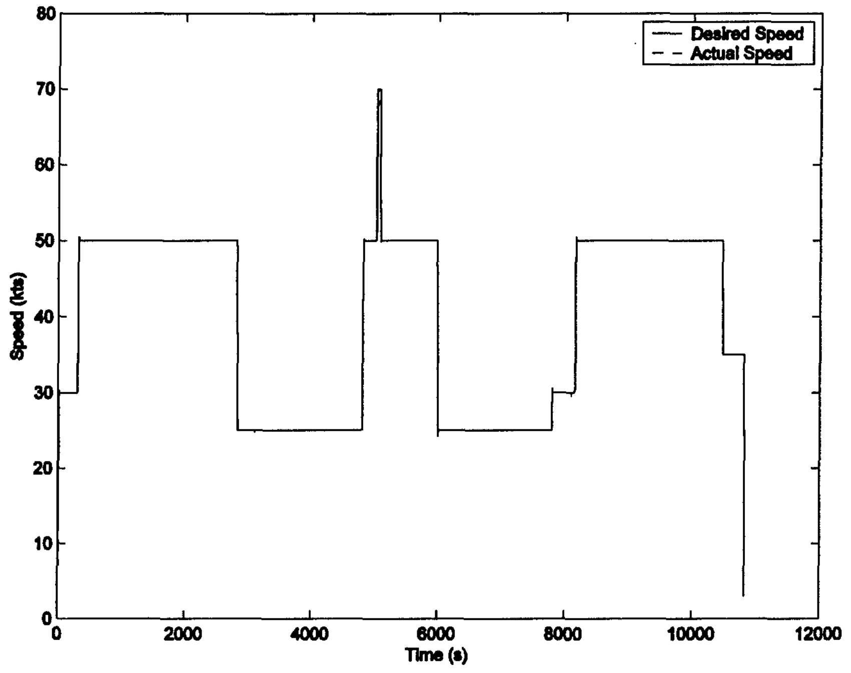


Figure 6-64: Desired and Actual UAV Speed, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Three-Input CMAC Controller (L=14), Charge-Sustaining

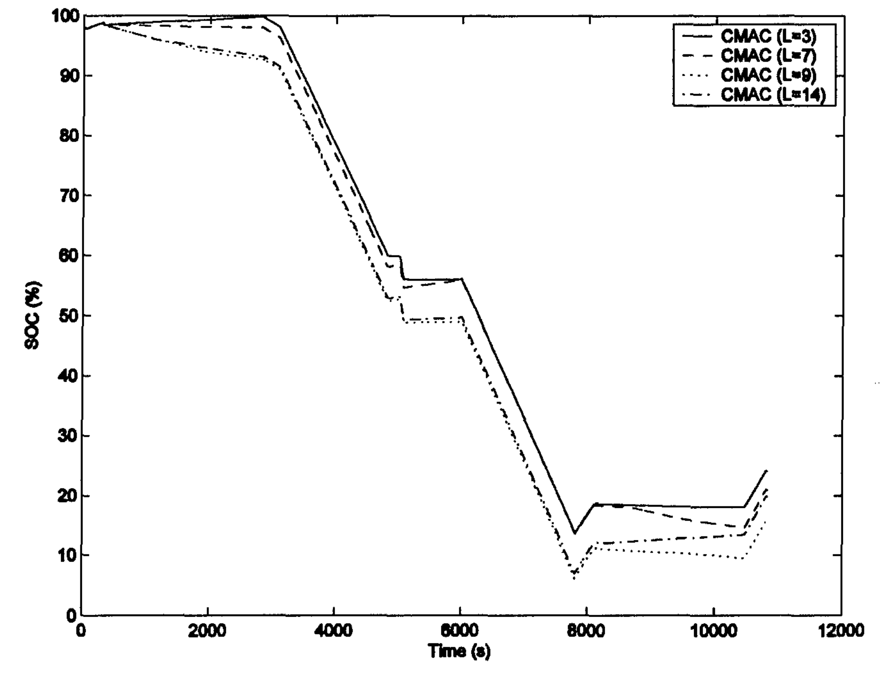


Figure 6-65: Battery SOC, Three-Hour Mission, Four-Stroke, HEUAV Configuration, Three-Input CMAC Controllers, Charge-Sustaining

The power and energy use for the HEUAV using the different controllers is shown in Table 6-11. The power consumption is listed for each mission segment. The fuel use is converted to an electricity equivalent. The energy use shows that the original configuration uses the most energy. The energy use for the HEUAV using the rule-based controller is 22% less than the original configuration. The optimization algorithm result (i.e. LUT) uses 27% less energy than the original configuration and 6.10% less than the HEUAV with the rule-based controller. For the rule-based controller, the amount of recharging is much greater at the end of the mission than at the beginning of the mission. The CMAC approximations are close to the optimization results and show that a generalization factor up to 14 can be used. Values of L=3, 7, 9, or 14 still provide reasonable results and use less energy as compared to the rule-based controller. In summary, a CMAC controller (L=14) would save two orders of magnitude on memory and would use 5.7% less energy than the rule-based controller.

Table 6-11: Power Consumption and Energy Use Summary for Three-Hour Flight Profile, Four-Stroke Engine

Mission Segment	**Engine Only**	**Rule-Based**	**Three-Input CMAC, Charge-Sustaining**				
			LUT	L=3	L=7	L=9	L=14
Power Consumption							
Take-Off (W)	10074	7318	6979	6985	6961	6945	6995
Climb (W)	3084	2438	2651	2652	2656	2647	2654
Cruise (W)	4127	4466	3549	3545	3632	3548	3568
Endurance (W)	1755	239.8	239.7	239.7	239.7	239.8	239.7
Max Speed (W)	10128	7140	7459	7461	7460	7456	7456
Descent (W)	1796	1921	1834	1829	1855	1849	1871
Energy Use							
Fuel (g)	779.3	582.1	544.3	548.4	546.1	541.5	545.8
Fuel (kWh)	9.27	6.98	6.52	6.57	6.55	6.50	6.54
Electricity (kWh)	N/A	0.240	0.253	0.244	0.245	0.266	0.263
Total (kWh)	9.27	7.22	6.78	6.82	6.79	6.76	6.81

6.6.3: Cruise Mission Segment with Wind Turbulence

The two ISR missions include primarily steady state conditions. Transients exist between mission segments, but each mission segment assumes constant conditions such as a steady wind, constant speed, and constant altitude. This section will present results for a cruise mission segment with wind turbulence to illustrate the improvements of the three-input CMAC neural network over the rule-based controller during a mission with variable forces acting on the UAV. One of the proposed applications for the HEUAV is disaster monitoring such as the real-time observation of forest fires. Turbulence would be experienced during these conditions, so wind turbulence is included in the model to further understand the capabilities and usefulness of the thee-input CMAC controller for the HEUAV application.

The UAV is treated as a point mass in the MATLAB/Simulink model as described in Equation 4-5. The wind turbulence is treated as another force acting on the UAV as shown in the UAV model in Figure 4-13. The turbulence is described using the turbulence intensity, a probability density function, and an autocorrelation function. The turbulence intensity (TI) is described as $TI=\sigma/U$ where σ is the standard deviation and U is the average speed of the wind [155, 156]. For wind turbulence with an average speed of 10 m/s and a TI of 0.3 (mountains), the standard deviation is 3.0. Manwell et al. and Rohatgi et al. state that wind turbulence can be modeled using a well-known Gaussian probability distribution function [155, 156]. The distribution function gives a measure of the likelihood of the wind speed of the turbulence, but it does not describe the speed based on what is has been in the past. The autocorrelation function relates the wind speed at an instant in time to a previous wind speed as given by a time lag. An example of the autocorrelation function for wind turbulence with a mean speed of 10 m/s, TI of 0.3, and a maximum time lag of 100 s is shown in Figure 6-66. The autocorrelation function, R, for this data can be described as:

$$R = \frac{1}{\sigma^2 (N_S - r)} \sum_{i=1}^{N_S - r} u_i u_{i+r} \qquad \textbf{(6-3)}$$

where N_S is the number of samples, r is the time lag in seconds, and u is the wind speed [155]. The sample data for the wind was generated using software developed by the University of Massachusetts for the Renewable Energy Research Laboratory [157]. For this sample data, wind is not correlated to wind that occurred greater than 100 s in the past.

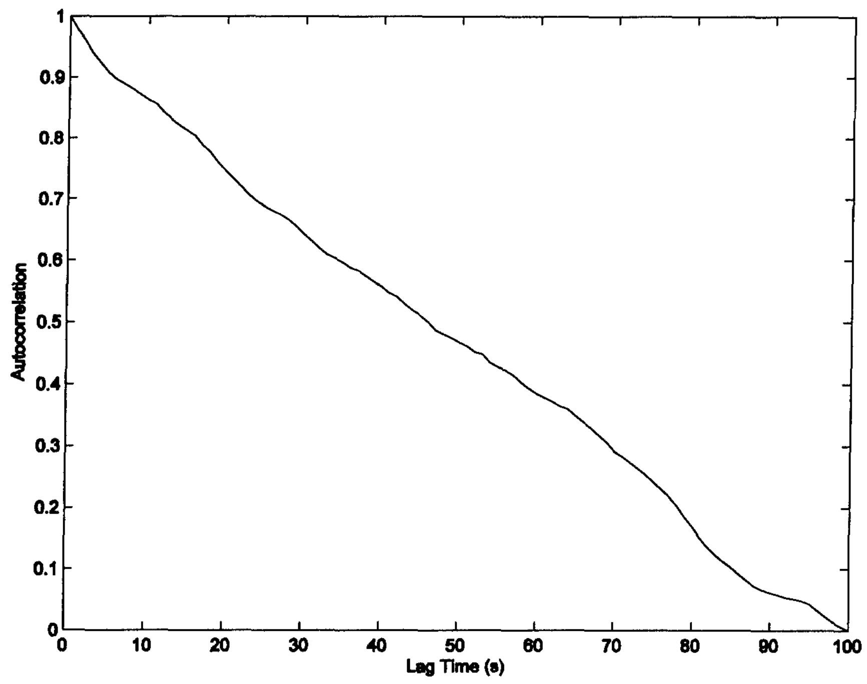


Figure 6-66: Autocorrelation of Sample Wind Turbulence

The forces acting on the HEUAV need to be related to the changes in wind speed. At a cruise speed of 50 kts, a change in wind speed of 1 m/s corresponds to a change in force of approximately 1.5 N acting on the UAV that was designed in Chapter 5. Therefore, 1 m/s of wind turbulence is approximately equivalent to a change in force of 1.5 N at cruise speed. The change in wind speed was equated to the changes in force at the cruise speed and used to vary the forces acting on the UAV during a cruise mission segment. A plot of the wind turbulence speed is shown in Figure 6-67. A 100 s turbulence sample is repeated ten times to produce the entire mission segment which lasts over fifteen minutes.

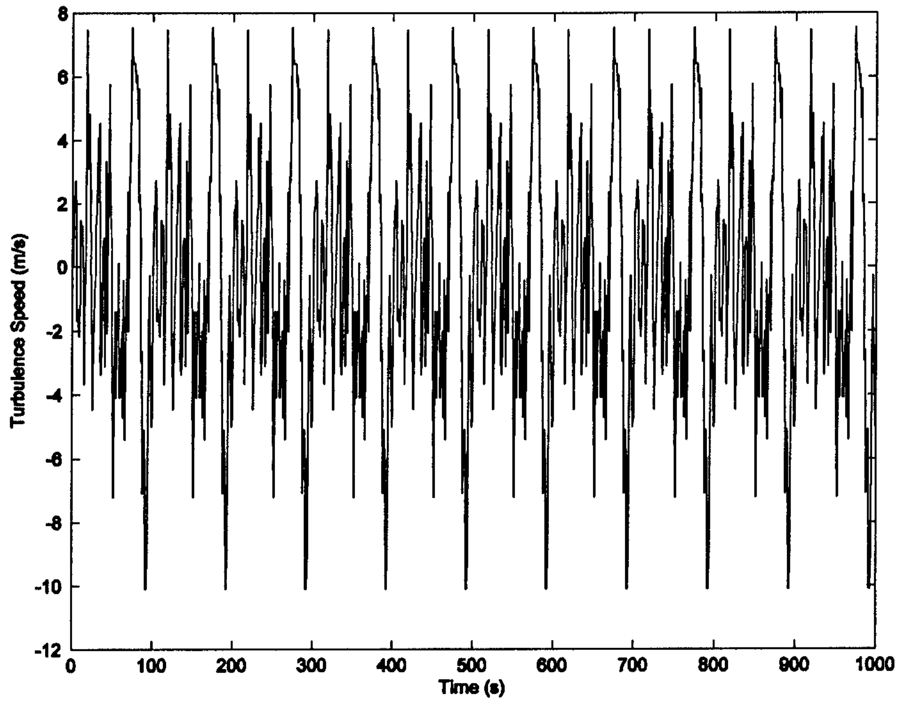


Figure 6-67: Turbulence Speed for Cruise Mission Segment

Simulations were completed for the four-stroke engine using the original configuration, the rule-based controller, and the three-input CMAC controller. The

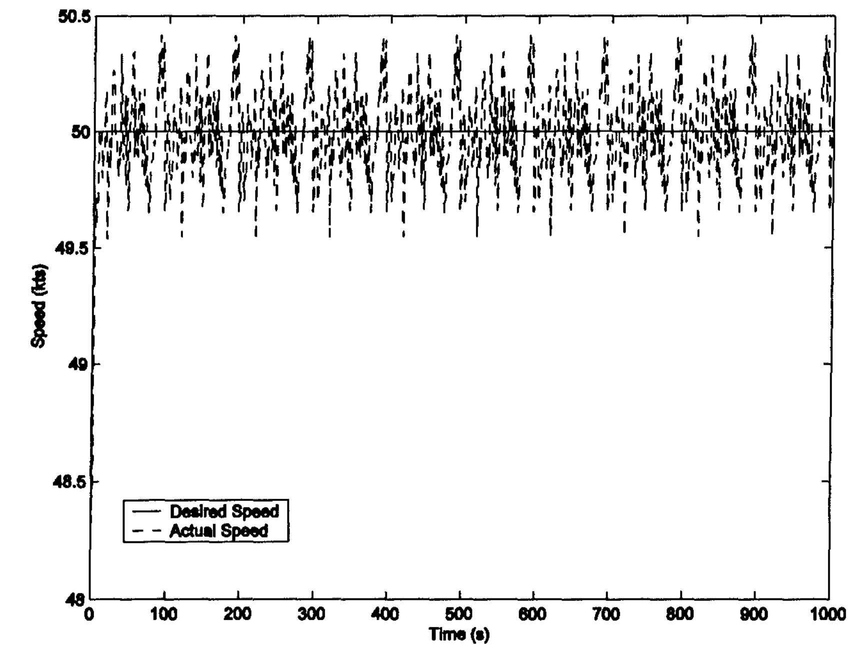


Figure 6-68: Actual and Desired UAV Speed, Cruise Mission Segment with Wind Turbulence, Four-Stroke, HEUAV Configuration, CMAC Controller (L=14), Charge-Sustaining

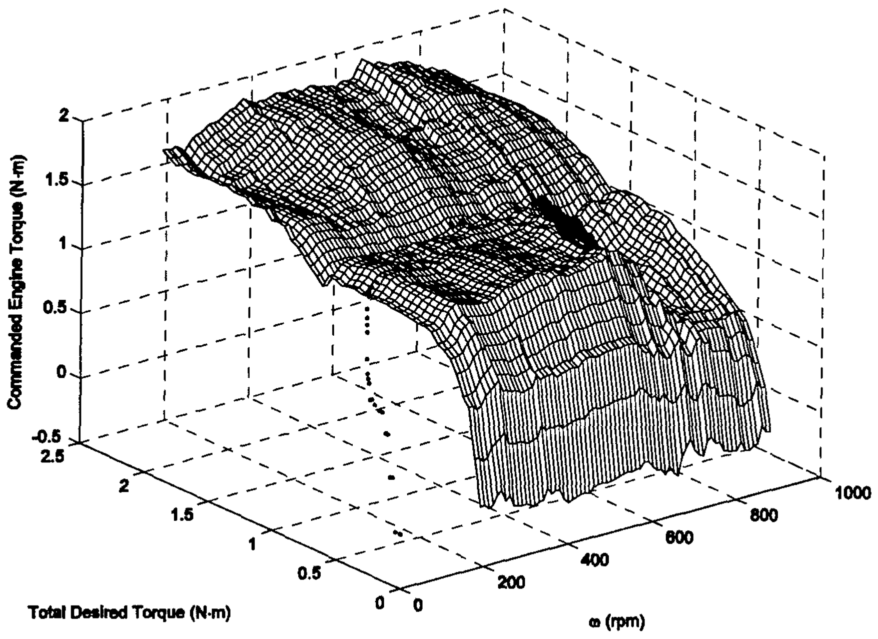


Figure 6-69: Engine Torque Control Surface and Operating Points, Cruise Mission Segment with Wind Turbulence, Four-Stroke, HEUAV Configuration, CMAC Controller (L=14), Charge-Sustaining

charge-sustaining control surfaces or algorithms were used in the simulations. The battery SOC was set to 50% at the beginning of the simulations. A sample speed plot for the UAV for the CMAC controller (L=14) is shown in Figure 6-68. The pilot/operator model keeps the UAV within 0.5 kts of the desired cruise speed of 50 kts. A plot of the commanded torque is shown in Figure 6-69 for the CMAC controller (L=14). The location of the points for the commanded torque is in a much larger area of the input space as compared to the flight profiles with constant forces acting on the UAV. The other simulations that included the wind turbulence had similar results.

The cruise mission segment with wind turbulence shows the improvement of the three-input CMAC controller over the rule-based controller and original configuration (see Table 6-12). The rule-based controller uses 12% less energy than the original configuration, and the CMAC controller (L=14) uses 15% less energy than the engine-only configuration. The CMAC controller algorithm (L=14) uses less energy (3.2%) than the rule-based controller which illustrates the advantage of the neural network controller over the rule-based controller for this type of mission segment. For other types of disturbances or varying forces acting on the UAV, similar results would be expected.

Table 6-12: Energy Use Summary for Cruise Mission Segment with Wind Turbulence, Four-Stroke Engine, Charge-Sustaining

Mission Segment	Engine Only	Rule-Based	Three-Input CMAC Controller	
			LUT	L=14
Fuel (g)	94.0	82.3	80.0	79.2
Fuel (kWh)	1.12	0.986	0.959	0.949
Electricity (kWh)	N/A	0.00186	0.00209	0.00665
Total (kWh)	1.12	0.988	0.961	0.956

6.7 Memory Savings

For each CMAC approximation to the engine torque control surface, the memory savings as compared to a look-up table (LUT) were listed. The number of weights was also included. If each weight requires 4 bytes (32 bits), then each approximation requires memory space, in bytes, four times that of the number of weights required. For example, the two-input controller requires 3,570 entries or 14,280 bytes for a look-up table but would require only 9.89 times less memory or 1,444 bytes for the CMAC approximation. Clearly, for the two-input controller, the look-up table or the original rules would be sufficient. However, as the complexity of the controller increases, the memory savings is more significant. For the three-input controller, the LUT requires 178,500 entries or 714 kbytes (0.714 MB). If the same embedded controller is needed for the flight control system, navigation, telemetry, or the payload, then the memory savings is significant. For the three-input CMAC controller with a generalization factor of L=14, the memory savings is a factor of 105.2. This means that instead of 0.714 MB, the CMAC controller only requires approximately 7 kbytes of memory to achieve approximately the same result as a large LUT that would represent the optimization results from the optimization algorithm.

6.8 Conclusions

This chapter presented an optimization algorithm and the design of the CMAC neural network controller used to approximate the optimization results. The optimization algorithm is flexible and permits the operator to generate CD or CS surfaces depending on the mission. The three-input CMAC controller consistently used less electrical energy than the original configuration or the HEUAV with the rule-based controller.

Chapter 7: Conclusions and Recommendations

7.1 Summary

Parallel hybrid-electric propulsion systems would be beneficial for small unmanned aerial vehicles (UAVs) used for military, homeland security, and disaster-monitoring missions. The benefits, due to the hybrid and electric-only modes, include increased endurance time and greater range as compared to electric-powered UAVs and a stealth mode not available with gasoline-powered UAVs. The dissertation research presented a conceptual design for the parallel hybrid-electric propulsion system and the associated small UAV, an algorithm for the instantaneous optimization of the energy use on-board the UAV, and the application of a CMAC artificial neural network to approximate the engine torque control surfaces obtained from the optimization algorithm. The conceptual design is a two-point design that includes an engine primarily sized for cruise speed and a battery pack and electric motor primarily sized for a slower endurance speed. The optimization algorithm that was developed can be used to generate charge-sustaining or charge-depletion control surfaces depending on the length of the intended mission. The various control surfaces were implemented in several flight profiles to illustrate and compare the usefulness of the different controllers. The CMAC approximations to the control surfaces reduce the memory requirement as compared to a look-up table (LUT) by one to two orders of magnitude. The CMAC controllers also prevent the need to compute a hyper-plane or complicated logic every time step. Although the CMAC neural network was trained off-line, the associative memory weights could be directly transferred to an embedded controller. Another option would be to program the insight and knowledge gained into a rule-based controller. For example, instead of using a proportional-derivative (PD) controller for charge-sustaining operation, a constant power output from the engine could be programmed into the rule-based controller. The research presented in the dissertation has generated several questions which could provide interesting research topics.

7.2 Potential Future Research

7.2.1 Dynamometer Testing of Advanced Propulsion Systems

As UAVs are used for more applications, computerized dynamometer test stands will be needed to test more fuel efficient internal combustion engines and propulsion systems. The commercially available dynamometers are expensive but a relatively low cost dynamometer, such as the one developed at Oklahoma State University, would be useful for the testing of UAV propulsion systems [158]. The dynamometer needs to be reliable and accurate with the capability to convert all data to standard day atmosphere to permit comparisons to simulation data. Parameters such as torque, thrust, rotational speed, fuel flow, voltage, and current need to be measured. Thrust and airspeed can be measured accurately if the propulsion system, including the propeller, is placed in a wind tunnel. The development of an advanced dynamometer test stand for UAV propulsion systems would provide an excellent research project for students.

7.2.2 Adaptive Scheme/Real-Time CMAC ANN Training

The CMAC neural network controller developed in this research is trained off-line. The weight values of the CMAC controller can then be directly transferred to the memory of an embedded microcontroller. If the torque output of the engine and electric motor could be measured or estimated, then an adaptive controller could be developed to optimize the energy use in real-time. As the atmospheric conditions change, the torque output of the engine could be varied to optimize the energy use in real-time to increase the range of the UAV.

7.2.3 Higher-Order Basis Functions for the CMAC ANN

The basis functions currently used in the associative memory of the CMAC neural network are binary. A stair-step output is produced from the CMAC controller. A filter could be placed on the output of the controller to smooth out the commands. The binary functions could also be replaced with linear or quadratic functions to produce a smoother function approximation if required in a hardware implementation. The convergence rate is also faster if higher-order basis functions are used but the training time is longer and the memory requirements are larger.

7.2.4 CMAC ANN Input Space Adaptation

Depending on the resulting hyper-plane surface, a uniform input space for the CMAC controller may not be sufficient. Higher resolution may be needed for the surface in certain areas, especially where discontinuities exist. Limited research has been completed in this area. A nonlinear placement strategy can account for the discontinuities in the training data. By placing multiple nodes in the same location, discontinuities can be better modeled.

7.2.5 Variable Pitch Propeller

A fixed pitch propeller was used in the HEUAV model, but as with most propellers, it operates at its peak efficiency over a short range of advance ratio. For the HEUAV, the advance ratio was similar at cruise and endurance speed. However, at other speeds that may be required for various missions, the propeller may not operate near its maximum efficiency. A variable pitch propeller permits a high efficiency over a wider range of advance ratio. The desired pitch could be an output from the propulsion system controller.

Appendix: MATLAB and C++ Code

A.1 Conceptual Design MATLAB Code

The conceptual design of the parallel hybrid-electric propulsion system for a small UAV was explained in Chapter 5. This section includes the MATLAB code developed to optimize the size of the wing and the components of the propulsion system.

```
% Algorithm for Sizing Hybrid-Electric UAV

% References:
% Anderson, Intro to Flight, 4th Edition
% Anderson, Aircraft Performance and Design
% Raymer, Aircraft Design: A Conceptual Approach, 2nd Edition

% Clear Workspace
close all;
clear all;

% UAV and Component Parameters
uav_Cdo_ref=0.035; % Zero-Lift Drag Coefficient for reference S, V=10 m/s
uav_S_ref=1.5; % Wing Area Reference for the reference Cdo (m^2)
uav_e=0.85; % Oswald Efficiency Factor
prop_m=0.17; % Prop Mass (kg), 20x8
prop_eff=0.75; % Typical Prop Efficiency during climb, cruise, endurance
bat_ED=100; % Battery Energy Density (Wh/kg), assume lithium-ion
EM_eff=0.85; % Typical EM Efficiency
EM_overtrq=2; % EM Over-Torque Factor
clutch_m=0.25; %  Clutch Mass for HEUAV (kg)
SFC_cruise=2.45E-6; % Specific Fuel Consumption (900 g/kWhr*9.81 m/s^2=2.45E-6 N/Ws), Anderson-
P&D, pg 154
SFC_endure=3.27E-6; % Specific Fuel Consumption (1200 g/kWhr*9.81 m/s^2=3.27E-6 N/Ws),
Anderson-P&D, pg 154

% Design Parameters
des_uav_m=13.6; % UAV Mass (kg), 13.6 kg=>30 lbs
des_pay_m=2.72; % Payload Mass (kg), 2.72 kg=>6 lbs
des_pay_P=75; % Payload Power (W)
des_uav_S=1.5; % Wing Area of Design (m^2)
des_uav_AR=15; % AR of Design
WF_empty=0.63; % Weight Fraction for the Empty Weight, Anderson-P&D, pg 400, Aerosonde UAV

% Performance Parameters
perf_tendure=3600; % Time for Endurance (s)
perf_tcruise=3600; % Time for Cruise, one-way (s)
perf_ROC=2.032; % Rate-of-Climb (m/s), 2=>400 ft/min
perf_Vstall=10.3; % Stall Velocity (m/s), 10.3 m/s=>20 kts
perf_Vendure=12.9; % Endurance Velocity (m/s), 12.9 m/s=>25 kts
perf_Vcruise=25.7; % Cruise Velocity (m/s), 25.7 m/s=>50 kts
perf_Vmax=30.9; % Max Velocity (m/s), 30.9 m/s=>60 kts
sdmin=2.57; % Minimum Speed Delta between Stall and Endurance Speeds (m/s), 2.57 m/s=>5 kts
```

```
 perf_s_TO=25; % Take-Off Distance (m)
 perf_ceiling=4572; % Service Ceiling (m), 15000 ft

 % Constants and Preliminary Calculations
 g=9.81; % Acceleration of Gravity (m/s^2)
 air_viscosity=2E-5; % Air Viscosity (N*s/m^2), FE Text-pg 11-2
 msec2kts=1.944; % Unit Conversion (m/s->kts)
 kts2msec=0.5144; % Unit Conversion (kts->m/s)
 W2hp=1.34/1000; % Unit Conversion (W->hp)
 hp2W=0.75*1000; % Unit Conversion (hp->W)
 uav_W=des_uav_m*g; % Weight of UAV (N)
 prop_W=prop_m*g; % Prop Weight (N)
 clutch_W=clutch_m*g; % Clutch Weight (N)

 % Select Altitude for the Calculations
 rhoSL=1.225; % Air Density (kg/m^3), Sea Level=>1.225
 disp(' ');
 disp('Select Altitude for the Calculations:');
 disp('Note:  All Take-off Calculations are Done at Sea Level.');
 disp('  1:  Sea Level');
 disp('  2:  5000 ft (1524 m)');
 disp('  3:  10000 ft (3048 m)');
 disp('  4:  15000 ft (4572 m)');
 profile=input('Enter your selection:  ');
 switch profile
    case 1
       rho=1.225;
    case 2
       rho=1.056;
    case 3
       rho=0.9048;
    case 4
       rho=0.7711;
    case 5
       disp('Not a valid selection.');
end

% Optimize the Design
disp(' ');
disp('Select optimization/solution routing:');
disp('  1:  Harmon optimization equations');
disp('  2:  Chattot optimization equations');
profile=input('Enter your selection:  ');
switch profile
   case 1
      % Minimize Endurance Power (Nonlinear Equation) with Equality Constraints
      x0=[100; 12; 1.2; 10; 11; 1000; 0.035]; % Initial values for x
      lb=[90; 8; 1; 5; 5; 500; 0.025]; % Lower Bound for variables
      ub=[200; 20; 1.25; 20; 20; 2500; 0.045]; % Upper Bound for variables
      disp(' ');
      disp('Minimizing EM Power, Optimizing W/S, AR, Clmax, Vstall, Vendure, ICE Power, Cdo:)');
      disp(' ');
      options=optimset('LargeScale','off','Display','final','MaxIter',15000,'MaxFunEvals',75000);
[x,fval,exitflag,output]=fmincon(@SizeEqMinPR,x0,[],[],[],[],lb,ub,@SizeEqCon,options,uav_W,uav_Cdo
_ref,uav_S_ref,uav_e,prop_eff,perf_Vstall,perf_Vendure,perf_Vcruise,perf_Vmax,rho,sdmin);
      EM_P_W=fval; % EM Power (W)
    disp(' ');
    disp('Optimization Completed:');
    disp(['Exitflag (>0 if Converged, =0 if Max Iterations, <0 if No Convergence):  ', num2str(exitflag)])
    output
    output.algorithm
    disp(' ');
    disp('Solution Computed from Optimization (Harmon):');
    disp(['W/S (N/m^2):  ', num2str(x(1))]);
    disp(['Aspect Ratio:  ', num2str(x(2))]);
    disp(['Clmax:  ', num2str(x(3))]);
    disp(['Vstall (kts):  ', num2str(msec2kts*x(4))]);
    disp(['Vendure (kts):  ', num2str(msec2kts*x(5))]);
    disp(['EM Power (W):  ', num2str(EM_P_W)]);
    disp(['ICE Power (W):  ', num2str(x(6))]);
    disp(['Zero-Lift Drag Coefficient:  ', num2str(x(7))]);

    % Assign Variables
    WLstall=x(1); % Wing Loading for Stall (N/m^2), Anderson-P&D, Eqn 8.26
    uav_WL=WLstall; % Wing Loading (N/m^2)
    uav_AR=x(2); % Aspect Ratio
    uav_Clmax=x(3); % Max Cl during Cruise
    uav_ClmaxTO=uav_Clmax; % Max Cl during Take-Off, assume no flaps
    Vstall=x(4);
    Vendure=x(5); % Endurance Velocity (m/s)
    ICE_P_W=x(6); % ICE Power (W)
    uav_Cdo=x(7); % Zero-Lift Drag Coefficient

    % Compare Equation Values to Solution Values
    disp(' ');
    disp('Percent Error Between Parameter Values and Equation Values:');
    disp(['W/S Error (%):  ', num2str(100*(x(1)-0.5*rho*Vstall^2*x(3))/x(1))]) ;
    disp(['AR Error (%):  ', num2str(100*(x(2)-
(2*x(1)/(rho*sqrt(3*uav_Cdo*pi*uav_e)*x(5)^2))^2)/x(2))]);
    disp(['Vstall Error (%):  ', num2str(100*(x(4)-sqrt(2*x(1)/(rho*x(3))))/x(4))]);
    disp(['Vendure Error (%):  ', num2str(100*(x(5)-
sqrt(2*x(1)*sqrt(1/(3*uav_Cdo*pi*uav_e*uav_AR))/rho))/x(5))]);
    disp(['EM Power Error (%):  ', num2str(100*(EM_P_W*prop_eff-
uav_W*sqrt(2*x(1)/rho)*4*uav_Cdo/(3*uav_Cdo*pi*uav_e*x(2))^0.75)/(prop_eff*EM_P_W))]);
    disp(['ICE Power Error (%):  ', num2str(100*(prop_eff*0.8*x(6)-
(0.5*rho*perf_Vcruise^3*uav_W*uav_Cdo/x(1)+2*perf_Vcruise*uav_W*x(1)/(rho*perf_Vcruise^2*pi*u
av_e*x(2))))/(prop_eff*0.8*x(6)))]);
    disp(['Zero-Lift Drag Coefficient Error (%):  ', num2str(100*(x(7)-
uav_Cdo_ref*uav_S_ref*x(1)*(10/x(5))^0.2/uav_W)/x(7))]);

  case 2
    % Minimize Endurance Power (Nonlinear Equation) with Equality Constraints
    x0=[100; 12; 1.2; 10; 11; 1000; 0.035]; % Initial values for x
    lb=[90; 8; 1; 5; 5; 500; 0.025]; % Lower Bound for variables
    ub=[200; 20; 1.25; 20; 20; 2500; 0.045]; % Upper Bound for variables
    disp(' ');
    disp('Minimizing EM Power, Optimizing W/S, AR, Clmax, Vstall, Vendure, ICE Power, & Cdo:)');
    disp(' ');
    options=optimset('LargeScale','off','Display','final','MaxIter',15000,'MaxFunEvals',75000);
[x,fval,exitflag,output]=fmincon(@SizeEqMinPR2,x0,[],[],[],[],lb,ub,@SizeEqCon2,options,uav_W,uav_C
do_ref,uav_S_ref,uav_e,prop_eff,perf_Vstall,perf_Vendure,perf_Vcruise,perf_Vmax,rho,sdmin);
    EM_P_W=fval; % EM Power (W)
```

```
        disp(' ');
        disp('Optimization Completed:');
        disp(['Exitflag (>0 if Converged, =0 if Max Iterations, <0 if No Convergence):  ', num2str(exitflag)]);
        output
        output.algorithm
        disp(' ');
        disp('Solution Computed from Optimization (Chattot):');
        disp(['W/S (N/m^2):  ', num2str(x(1))]);
        disp(['Aspect Ratio:  ', num2str(x(2))]);
        disp(['Clmax:  ', num2str(x(3))]);
        disp(['Vstall (kts):  ', num2str(msec2kts*x(4))]);
        disp(['Vendure (kts):  ', num2str(msec2kts*x(5))]);
        disp(['EM Power (W):  ', num2str(EM_P_W)]);
        disp(['ICE Power (W):  ', num2str(x(6))]);
        disp(['Zero-Lift Drag Coefficient:  ', num2str(x(7))]);

        % Assign Variables
        WLstall=x(1); % Wing Loading for Stall (N/m^2), Anderson-P&D, Eqn 8.26
        uav_WL=WLstall; % Wing Loading (N/m^2)
        uav_AR=x(2); % Aspect Ratio
        uav_Clmax=x(3); % Max Cl during Cruise
        uav_ClmaxTO=uav_Clmax; % Max Cl during Take-Off, assume no flaps
        Vstall=x(4); % Stall Velocity (m/s)
        Vendure=x(5); % Endurance Velocity (m/s)
        ICE_P_W=x(6); % ICE Power (W)
        uav_Cdo=x(7); % Zero-Lift Drag Coefficient

        % Compare Equation Values to Solution Values
        disp(' ');
        disp('Percent Error Between Parameter Values and Equation Values:');
        disp(['W/S Error (%):  ', num2str(100*(x(1)-0.5*rho*x(4)^2*x(3))/x(1))]);
        disp(['AR Error (%):  ', num2str(100*(x(2)-
((x(4)/x(5))^2*x(3)/sqrt(3*pi*uav_e*uav_Cdo))^2)/x(2))]);
        disp(['Vstall Error (%):  ', num2str(100*(x(4)-sqrt(2*x(1)/(rho*x(3))))/x(4))]);
        disp(['Vendure Error (%):  ', num2str(100*(x(5)-
sqrt(2*x(1)*sqrt(1/(3*uav_Cdo*pi*uav_e*uav_AR))/rho))/x(5))]);
        disp(['EM Power Error (%):  ', num2str(100*(EM_P_W-
4*uav_Cdo*uav_W*x(5)*(x(5)/x(4))^2/(prop_eff*x(3)))/EM_P_W)]);
        disp(['ICE Power Error (%):  ', num2str(100*(prop_eff*0.8*x(6)-
(1+3*(x(5)/perf_Vcruise)^4)*(perf_Vcruise/x(4))^2*perf_Vcruise*uav_W*uav_Cdo/x(3))/(prop_eff*0.8*x
(6)))]);
        disp(['Zero-Lift Drag Coefficient Error (%):  ', num2str(100*(x(7)-
uav_Cdo_ref*uav_S_ref*x(1)*(10/x(5))^0.2/uav_W)/x(7))]);

        % Plot ICE Power vs. Vstall and Clmax
        sd=Vendure-Vstall; % Speed Delta between Stall and Endurance Speeds (m/s), 2.57 m/s=>5 kts
        Vstall_range=[7:0.1:15]; % Range of Stall Velocities (m/s)
        Clmax_range=[1:0.01:2.5]; % Range of Max Lift Coefficients
        figure;
        [V,Cl]=meshgrid(Vstall_range,Clmax_range);
ICE_P1=(1*ones(size(V))+3*((V+sd*ones(size(V)))./perf_Vcruise).^4).*(perf_Vcruise./V).^2*perf_Vcruis
e*uav_W*uav_Cdo./(Cl*prop_eff*0.8);
        [C,h]=contour(V,Cl,ICE_P1,'k');
        clabel(C,h);
        hold on;
ICE_P2=(ones(size(V))+3*((V+sd*ones(size(V)))./perf_Vmax).^4).*(perf_Vmax./V).^2*perf_Vmax*uav_
```

```
W*uav_Cdo./(Cl*prop_eff)-
4*EM_overtrq*uav_Cdo*uav_W*(V+sd*ones(size(V))).*((V+sd*ones(size(V)))./V).^2./(prop_eff*Cl);
    [C,h]=contour(V,Cl,ICE_P2,'r:');
    clabel(C,h);
    [C,h]=contour(V,Cl,ICE_P1-ICE_P2,[0 -5000],'b');
    clabel(C,h);
    plot(Vstall,uav_Clmax,'bo');
    xlabel('Stall Velocity (m/s)'); ylabel('Max Lift Coefficient, Cl');
    title({'ICE Power Required';'| Cruise (W)  : Max Velocity (W)'});
    %print -depsc2 -tiff -r600 UAV_ICE_P_Vstall_Cl

  case 3
    disp('Not a valid selection.');
end

% Plot Cf vs. Reynolds Number
Re1=[1E3:1000:1E6]; % Range of Reynolds Number
Re2=[1E5:1000:1E8]; % Range of Reynolds Number
Cflam=1.328./sqrt(Re1); % Cf for Laminar Flow, Raymer, Eqn 12.25
Cfturb=0.455./(log10(Re2)).^2.58; % Cf for Turbulent Flow, Raymer, Eqn 12.27
figure;
loglog(Re1,Cflam,'b:');
hold on;
loglog(Re2,Cfturb);
xlabel('Reynolds Number, Re'); ylabel('Cf');
title('Flat-Plate Skin-Friction Coefficient vs. Reynolds Number');
legend('Turbulent','Laminar');
grid on;
%print -depsc2 -tiff -r600 UAV_Cf_Re

% Determine Wing Area and Wing Loading
uav_S=uav_W/WLstall; % Calculated Wing Area of UAV based on Wing Loading for Stall (m^2)
if (uav_S>des_uav_S)
  disp(' ');
  disp('S is larger than desired due to Wing Loading for Stall->Increase CLmax, Vstall, or S');
  disp(['Desired S (m^2):  ', num2str(des_uav_S)]);
  disp(['Calculated S (m^2):  ', num2str(uav_S)]);
end

% Calculate Endurance Parameter and L/D Ratio
uav_K=1/(pi*uav_e*uav_AR); % Constant
uav_Cl_15_Cd=(3*uav_Cdo/uav_K).^0.75/(4*uav_Cdo); % Endurance Parameter (Cl^1.5/Cd), Anderson-
Flight, Eqn 6.87
uav_Cl_Cd=sqrt(uav_Cdo/uav_K)/(2*uav_Cdo); % Lift-to-Drag Ratio (Cl/Cd), Anderson-Flight, Eqn 6.85

% Check Endurance Velocity and Stall Velocity
disp(' ');
disp(['Difference Between Desired and Actual Endurance Velocity (kts):  ',
num2str(msec2kts*(perf_Vendure-Vendure))]);
disp(['Difference Between Desired and Actual Stall Velocity (kts):  ', num2str(msec2kts*(perf_Vstall-
Vstall))]);
disp(['Difference Between Endurance Velocity and Stall Velocity (kts):  ', num2str(msec2kts*(Vendure-
Vstall))]);

% Wing Geometry Calculations
LDmax=sqrt(1/(4*uav_K*uav_Cdo)); % Maximum Lift-to-Drag Ratio, Anderson-P&D, Eqn 5.24
```

```
V_LDmax=sqrt(2*uav_WL/rho*sqrt(uav_K/uav_Cdo)); % Velocity at Max L/D (m/s)
design_Cl_endure=2/(rho*Vendure^2)*(uav_WL); % Design Lift Coefficient, Raymer, Eqn 4.5
design_Cl_cruise=2/(rho*perf_Vcruise^2)*(uav_WL); % Design Lift Coefficient, Raymer, Eqn 4.5
uav_wing_span=sqrt(uav_S*uav_AR); % Wing Span (m)
uav_wing_chord=uav_S/uav_wing_span; % Wing Chord (m)
uav_wing_Reynolds_Vendure=rho*Vendure*uav_wing_chord/air_viscosity; % Reynolds number of Wing
for Endurance
uav_wing_Reynolds_Vcruise=rho*perf_Vcruise*uav_wing_chord/air_viscosity; % Reynolds number of
Wing for Cruise
uav_wing_Reynolds_Vmax=rho*perf_Vmax*uav_wing_chord/air_viscosity; % Reynolds number of Wing
for Max Speed

% Plot to Show Relationship Between Vendure and PRendure vs. W/S and AR
uav_WL_range_ozft2=[10:0.5:50]; % Wing Loading (oz/ft^2)
uav_AR_range=[5:0.1:20]; % Aspect Ratio
K_range=1./(pi*uav_e*uav_AR_range); % Constant
[WL,AR]=meshgrid(uav_WL_range_ozft2,uav_AR_range);
Vendure_range=sqrt(2*3.0*WL./rho.*sqrt(1./(3*uav_Cdo*pi*uav_e*AR))); % Endurance Velocity (m/s),
Anderson-P&D, Eqn 5.41
figure;
[C,h]=contour(WL,AR,msec2kts*Vendure_range,2*[5 6 7 8 9 10 11 12 13 14 15],'k'); % Plot Endurance
Velocity (kts) vs. W/S and AR
clabel(C,h);
hold on;
Cl_15_Cd=(3*uav_Cdo*pi*uav_e*AR).^0.75./(4*uav_Cdo); % Endurance Parameter (Cl^1.5/Cd),
Anderson-Flight, Eqn 6.87
PR_Vendure_range=uav_W.*sqrt(2*3.0*WL./rho)./Cl_15_Cd; % Power Required for Endurance (W),
Anderson-P&D, Eqn 5.56
[C,h]=contour(WL,AR,PR_Vendure_range,[50 75 100 125 150 200],'k--');
clabel(C,h);
plot(0.333*uav_WL,uav_AR,'ko');
xlabel('Wing Loading, W/S (oz/ft^2)'); ylabel('Aspect Ratio, AR');
title('___ Endurance Speed (kts)  --- Power Required (W)  o Design Point');
%print -depsc2 -tiff -r600 UAV_PR_Vendure_WL_AR

% Plot Endurance Parameter and L/D Ratio vs. AR
uav_Cl_15_Cd_range=(3*uav_Cdo./K_range).^0.75./(4*uav_Cdo); % Endurance Parameter (Cl^1.5/Cd),
Anderson-Flight, Eqn 6.87
uav_Cl_Cd_range=sqrt(uav_Cdo./K_range)./(2*uav_Cdo); % Lift-to-Drag Ratio (Cl/Cd), Anderson-Flight,
Eqn 6.85
figure;
plot(uav_AR_range,uav_Cl_15_Cd_range,'b');
hold on;
plot(uav_AR_range,uav_Cl_Cd_range,'k');
plot(uav_AR,uav_Cl_15_Cd,'bo');
plot(uav_AR,uav_Cl_Cd,'ko');
xlabel('Aspect Ratio (AR)'); ylabel('Endurance Parameter or L/D Ratio');
title('Endurance Parameter and L/D Ratio vs. AR');
legend('Endurance Parameter, Cl^1^.^5/Cd','Lift-to-Drag Ratio, L/D','Design Points');
%print -depsc2 -tiff -r600 UAV_EndPar_LD_AR

% Determine Power Required for Endurance for 30 lb UAV
uav_S_range=[0.5:0.1:2]; % Wing Area (m^2)
[S,Cl_15_Cd]=meshgrid(uav_S_range,uav_Cl_15_Cd_range);
PRendure_range=sqrt(2*uav_W.^3./(rho.*S))./Cl_15_Cd; % Power Required (W), Anderson-P&D, Eqn
5.56
```

```
figure;
[C,h]=contour(S,Cl_15_Cd,PRendure_range);
clabel(C,h);
hold on;
plot(uav_S,uav_Cl_15_Cd,'o');
xlabel('Wing Area, S (m^2)'); ylabel('Endurance Parameter, Cl^1^.^5/Cd'); zlabel('Power Required (W)');
title('Endurance Power Required (W) for UAV mass=13.6 kg (30 lbs)');
legend('Design Point');
%print -depsc2 -tiff -r600 UAV_PRendure_S_EndPar

% Plot Power Required for Endurance and Battery Weight vs. Weight
W=[0:200]; % Range of Weights (N)
figure;
plot(W,sqrt(2*W.^3./(rho.*uav_S))./uav_Cl_15_Cd,'k'); % Assume Endurance Parameter is the current one
hold on;
plot(W,bat_ED*0.2*W*prop_eff*EM_eff/g-des_pay_P,'k:'); % Assume 20% of Weight is batteries
xlabel('Weight (N)'); ylabel('Power (W)');
title('Endurance Power Required and Battery Power Available');
legend('Power Required','Battery Power Available');
%print -depsc2 -tiff -r600 UAV_PRendure_Battery_Weight

% Plot Power Required for Endurance vs. Weight and Wing Area
W=[0:1:200]; % Range of Weights (N)
S=[0.5:0.1:2.5]; % Range of Wing Area (m^2)
figure;
[W,S]=meshgrid(W,S);
PRendure=sqrt(2*W.^3./(rho.*S))./uav_Cl_15_Cd; % Assume Endurance Parameter is the current one
[C,h]=contour(W,S,PRendure,[25 50 100 150 200 250 300 350 400],'k');
clabel(C,h);
hold on;
PAbat=bat_ED*0.2*W*prop_eff*EM_eff/g-des_pay_P*ones(size(W)); % Assume 20% of Weight is
batteries
[C,h]=contour(W,S,PAbat,'r:');
clabel(C,h);
[C,h]=contour(W,S,PRendure-PAbat,[0 -1000],'b');
clabel(C,h);
plot(uav_W,uav_S,'bo');
xlabel('Weight (N)'); ylabel('Wing Area, S (m^2)');
title({'Endurance Power Required and Battery Power Available (0.2 WF)';'| Power Required (W)  : Battery
Power Available (W)'});
%print -depsc2 -tiff -r600 UAV_PRendure_W_S

% Plot Power Required for Endurance vs. Weight and Endurance Parameter
W=[50:1:200]; % Range of Weights (N)
Cl_15_Cd=[10:0.1:20]; % Range of Wing Area (m^2)
figure;
[W,Cl_15_Cd]=meshgrid(W,Cl_15_Cd);
PRendure=sqrt(2*W.^3./(rho.*uav_S))./Cl_15_Cd; % Assume Wing Area is the current one
[C,h]=contour(W,Cl_15_Cd,PRendure,[25 50 100 150 200 300 400],'k');
clabel(C,h);
hold on;
PAbat=bat_ED*0.2*W*prop_eff*EM_eff/g-des_pay_P*ones(size(W)); % Assume 15% of Weight is
batteries
[C,h]=contour(W,Cl_15_Cd,PAbat,'k:');
clabel(C,h);
[C,h]=contour(W,Cl_15_Cd,PRendure-PAbat,[0 -1000],'k');
```

```
clabel(C,h);
plot(uav_W,uav_Cl_15_Cd,'ko');
xlabel('Weight (N)'); ylabel('Endurance Parameter (Cl^1.5/Cd)');
title({'Endurance Power Required and Battery Power Available (0.2 WF)';'| Power Required (W)  : Battery
Power Available (W)'});
%print -depsc2 -tiff -r600 UAV_Power_Weight_EndPar

% Plot Lift Coefficent, Cl, vs. Velocity
V=[Vstall:0.5:perf_Vmax]; % Range of Velocities (m/s)
Cl=uav_WL./(0.5*rho*V.^2); % Lift Coefficient, Anderson-Flight, Eqn 6.26
figure;
plot(msec2kts*V,Cl);
xlabel('Velocity (kts)'); ylabel('Lift Coefficient, Cl');
title('Lift Coefficient vs. Velocity');
%print -depsc2 -tiff -r600 UAV_LiftCoef_Speed

% Section for Original Configuration UAV
% Determine Power Required to Meet ROC Requirement
V=[Vstall-2.57:0.5:perf_Vmax]; % Range of Velocities (m/s)
Cl=uav_WL./(0.5*rho*V.^2); % Lift Coefficient, Anderson-Flight, Eqn 6.26
Cd=uav_Cdo+uav_K*Cl.^2; % Drag Coefficient, Anderson-Flight, pg 359
PR_SL=sqrt(2*uav_W^3*Cd.^2./(rho*uav_S*Cl.^3)); % PR for S&L (W), Anderson-Flight, Eqn 6.27
PR_climb=perf_ROC*uav_W*ones(size(Cd))+PR_SL; % PR for Climb (W), Anderson-Flight, Eqn 6.50
uav_PRmin_climb=min(PR_climb); % PR min in PR_climb vector (W)

% Plot Power Required to Climb
figure;
plot(msec2kts*V,PR_climb,'b');
xlabel('Velocity (kts)'); ylabel('Power (W)');
title('Power Required to Climb to Meet ROC Requirement (m=13.6 kg, 30 lbs)');
%print -depsc2 -tiff -r600 UAV_PR_ROC

% Determine Power Required to Meet Take-off Requirement
TR_TO=1.21*(uav_WL)/(g*rhoSL*uav_ClmaxTO*perf_s_TO/uav_W); % Thrust Required for Take-off
(N), Anderson-P&D, Eqn 6.95
Vtakeoff=1.2*sqrt(2*uav_WL/(rhoSL*uav_ClmaxTO)); % Velocity at Take-Off (m/s), 120% of Vstall,
Anderson-P&D, Eqn 5.67
uav_PR_TO=TR_TO*Vtakeoff; % Power Required for Take-off (W)

% Determine Power Required to Meet Max Velocity Requirement
Cl=uav_WL/(0.5*rho*perf_Vmax^2); % Lift Coefficient for Cruise, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Cruise, Anderson-P&D, Eqn 5.10
uav_PR_Vmax=sqrt(2*uav_W^3*Cd^2/(rho*uav_S*Cl^3)); % Power Required for Vmax (W), Anderson-
Flight, Eqn 6.27

% Determine Power Required to Meet Cruise Requirement
Cl=uav_WL/(0.5*rho*perf_Vcruise^2); % Lift Coefficient for Cruise, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Cruise, Anderson-P&D, Eqn 5.10
LDcruise=Cl/Cd; % L/D Ratio at Cruise
uav_PR_Vcruise=sqrt(2*uav_W^3*Cd^2/(rho*uav_S*Cl^3)); % Power Required for Vcruise (W),
Anderson-Flight, Eqn 6.27

% Determine Power Required to Meet Endurance Requirement
Cl=uav_WL/(0.5*rho*Vendure^2); % Lift Coefficient for Cruise, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Cruise, Anderson-P&D, Eqn 5.10
```

```
uav_PR_Vendure=sqrt(2*uav_W^3*Cd^2/(rho*uav_S*Cl^3)); % Power Required for Vendure (W),
Anderson-Flight, Eqn 6.27

% Place Power Requirements into a Vector
PR=[uav_PR_TO uav_PRmin_climb uav_PR_Vcruise uav_PR_Vendure uav_PR_Vmax]; % Power
Required for the Mission Segments (W)
ICE_P_W_org=max(PR/prop_eff)+1.25*des_pay_P; % Size of ICE to meet PR (W), includes payload
power and generator efficiency
ICE_P_hp_org=W2hp*ICE_P_W_org; % Size of ICE to meet PR (hp)
ICE_Size_in3_org=ICE_P_hp_org; % Size of ICE (in^3), assume 1 hp=1 in^3

% Mission Segment Weight Fractions for the Original Configuration
Wo=uav_W; % Desired UAV Weight (N)
WF_TO_org=0.97; % Weight Fraction for Take-off, Raymer-A/C Design, Section 3.4
WF_climb_org=0.985; % Weight Fraction for Climb, Raymer-A/C Design, Section 3.4
R=perf_Vcruise*perf_tcruise; % Range of UAV (m)
Cl=uav_WL/(0.5*rho*perf_Vcruise^2); % Lift Coefficient for Cruise, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Cruise, Anderson-P&D, Eqn 5.10
WF_cruise_org=exp(-R*SFC_cruise/(prop_eff*(Cl/Cd))); % Weight Fraction for Cruise, Derived from
Breguet Formula
Cl=uav_WL/(0.5*rho*Vendure^2); % Lift Coefficient for Endurance, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Endurance, Anderson-P&D, Eqn 5.10
WF_endure_org=exp(-Vendure*perf_tendure*SFC_endure/(prop_eff*(Cl/Cd))); % Weight Fraction for
Endurance, Raymer, Section 3.4
WF_landing_org=0.995; % Weight Fraction for Landing, Raymer, Section 3.4
WF_fuel_org=1.06*(1-
WF_TO_org*WF_climb_org*WF_cruise_org*WF_endure_org*WF_cruise_org*WF_landing_org); %
Weight Fraction for Fuel, Raymer, Section 3.4
pay_W_org=Wo*(1-WF_fuel_org-WF_empty); % Calculate Payload Weight (N)
pay_m_org=pay_W_org/g; % Payload Mass of Original UAV (kg)
WF_pay_org=pay_W_org/Wo; % Weight Fraction for the Payload
ICE_m_org=ICE_P_W_org/1000; % Engine Mass of Original UAV (kg), Assume 1.5 lb/hp or 1 kg/kW
fuel_m_org=WF_fuel_org*Wo/g; % Fuel mass of Original UAV (kg)
ess_m_org=0.5; % Mass of original Battery Pack or Generator (kg)
uav_empty_m_org=des_uav_m-fuel_m_org-pay_m_org; % Empty Mass of Original UAV (kg)
uav_glider_m=uav_empty_m_org-ICE_m_org-ess_m_org-prop_m; % Glider Mass of Original UAV (kg)
propulsion_m_org=ICE_m_org+ess_m_org+prop_m; % Mass of Propulsion System of Original UAV (kg)
WF_propulsion_org=propulsion_m_org*g/Wo; % Weight Fraction for Propulsion System for Original
UAV

% Plot Performance Requirements on a T/W and W/S Plot
uav_WL_range_ozft2=[10:0.5:50]; % Wing Loading (oz/ft^2)
uav_TW_range=[0.05:0.05:0.75]; % Thrust-to-Weight Ratio
[WL,TW]=meshgrid(uav_WL_range_ozft2,uav_TW_range);
PR_Vmax_range=TW*uav_W.*sqrt((TW.*3.0.*WL+3.0*WL.*sqrt(TW.^2-
4*uav_K*uav_Cdo))./(rho*uav_Cdo)); % Power Required for Vmax (W), Anderson-P&D, Eqn 5.50
figure
[C,h]=contour(WL,TW,PR_Vmax_range,'b'); % Plot Power Required for Vmax vs. T/W and W/S
clabel(C,h);
hold on;
PR_ROC_range=uav_W.*TW.*sqrt(2*3.0*WL/rho.*sqrt(uav_K/(3*uav_Cdo))); % Power Required for
Max ROC (W), Anderson-P&D, pg 266-267, 276
[C,h]=contour(WL,TW,PR_ROC_range,'g'); % Plot Power Required for Max ROC
clabel(C,h);
PR_TO_range=1.2*TW.*uav_W.*sqrt(2*3.0*WL/(rhoSL*uav_ClmaxTO)); % Power Required for Take-
off (W)
```

```
[C,h]=contour(WL,TW,PR_TO_range,'r'); % Plot Power Required for Take-off Roll (W)
clabel(C,h);
xlabel('Wing Loading, W/S (oz/ft^2)'); ylabel('Thrust-to-Weight Ratio, T/W'); zlabel('Power Required
(W)');
title({'Power Required (W) for Performance Parameters vs. T/W and W/S','Vmax-blue  ROCmax-green
Take-Off-red'});
%print -depsc2 -tiff -r600 UAV_Perf_TW_WL

% Plot Performance Requirements on a AR and W/S Plot
uav_WL_range_ozft2=[20:0.5:40]; % Wing Loading (oz/ft^2)
uav_AR_range=[5:0.1:15]; % Aspect Ratio
[WL,AR]=meshgrid(uav_WL_range_ozft2,uav_AR_range);
q_max=0.5*rho*perf_Vmax^2; % Dynamic pressure for Vmax
K_range=1./(pi*uav_e*AR); % Constant
PR_Vmax=perf_Vmax*(q_max*uav_S*uav_Cdo*ones(size(WL))+K_range.*uav_S.*(3.0*WL).^2/q_max
); % Power Required for Vmax (W), Anderson-P&D, Eqn 5.13
figure;
[C,h1]=contour(WL,AR,PR_Vmax,'b'); % Plot Power Required for Vmax (W)
clabel(C,h1);
hold on;
q_cruise=0.5*rho*perf_Vcruise^2; % Dynamic pressure for Vcruise
PR_Vcruise=perf_Vcruise*(q_cruise*uav_S*uav_Cdo*ones(size(WL))+K_range.*uav_S.*(3.0*WL).^2/q_
cruise); % Power Required for Cruise (W), Anderson-P&D, Eqn 5.13
[C,h2]=contour(WL,AR,PR_Vcruise,'c'); % Plot Power Required for Vcruise (W)
clabel(C,h2);
Cl_15_Cd=(3*uav_Cdo./K_range).^0.75./(4*uav_Cdo); % Endurance Parameter (Cl^1.5/Cd), Anderson-
Flight, Eqn 6.87
PR_Vendure=uav_W*sqrt(2*3.28*WL./rho)./Cl_15_Cd; % Power Required for Endurance (W), Anderson-
P&D, Eqn 5.56
[C,h3]=contour(WL,AR,PR_Vendure,'k');
clabel(C,h3);
VROC=sqrt(2*3.0*WL/rho.*sqrt(K_range/(3*uav_Cdo))); % Velocity for Max ROC, Anderson-P&D, Eqn
5.118
Cl=uav_WL./(0.5*rho*VROC.^2); % Lift Coefficient, Anderson-Flight, Eqn 6.26
Cd=uav_Cdo*ones(size(Cl))+uav_K*Cl.^2; % Drag Coefficient, Anderson-Flight, pg 359
PR_ROC=perf_ROC*uav_W*ones(size(Cd))+0.5*rho*VROC.^2.*uav_S.*Cd.*VROC; % Power
Required for Max ROC (W), Anderson-P&D, pg 276
[C,h4]=contour(WL,AR,PR_ROC,'g');
clabel(C,h4);
[C,h4]=contour(WL,AR,PR_Vmax-(EM_overtrq*PR_Vendure+1.25*PR_Vcruise),'r');
clabel(C,h4);
plot(0.333*uav_WL,uav_AR,'ro');
xlabel('Wing Loading, W/S (oz/ft^2)'); ylabel('Aspect Ratio, AR'); zlabel('Power Required (W)');
title({'Power Required (W) for Performance Parameters vs. AR and W/S','Vmax-blue  Vcruise-cyan
Vendurance-black  ROCmax-green'});
%print -depsc2 -tiff -r600 UAV_Perf_WL_AR

% Section for Hybrid-Electric UAV (HEUAV)
% Determine Size of EM and Mass of Batteries
%EM_P_W=uav_PR_Vendure/prop_eff; % Power of EM (W)
bat_m=1.25*EM_P_W*perf_tendure/(bat_ED*3600)+des_pay_P/bat_ED; % Mass of batteries for
PR_design (kg), includes payload power
bat_W=bat_m*g; % Weight of Batteries (N)
EM_m=EM_P_W*0.000603; % Mass of EM (Assume 1 lb/hp or 0.000603 kg/W)
EM_W=EM_m*g; % Weight of EM (N)
```

```
% Determine Power Required to Cruise and Size ICE
ICE_P_hp=W2hp*ICE_P_W; % Size of ICE to meet PR (hp)
ICE_Size_in3=ICE_P_hp; % Size of ICE (in^3), assume 1 hp=1 in^3

% Mission Segment Weight Fractions for HEUAV Configuration
Wo=uav_W; % Desired UAV Weight (N)
WF_EM=EM_W/Wo; % Weight Fraction of EM
WF_bat=bat_W/Wo; % Weight Fraction of batteries
WF_TO=0.97+0.015; % Weight Fraction for Take-off, 1/2 for Electric, Raymer-A/C Design, Section 3.4
WF_climb=0.985+0.0075; % Weight Fraction for Climb, 1/2 for Electric, Raymer-A/C Design, Section 3.4
R=perf_Vcruise*perf_tcruise; % Range of UAV (m)
Cl=uav_WL/(0.5*rho*perf_Vcruise^2); % Lift Coefficient for Cruise, Anderson-P&D, Eqn 5.11
Cd=uav_Cdo+uav_K*Cl^2; % Drag Coefficient for Cruise, Anderson-P&D, Eqn 5.10
WF_cruise=exp(-R*SFC_cruise/(prop_eff*(Cl/Cd))); % Weight Fraction for Cruise, Derived from Breguet
Formula
WF_endure=1; % Weight Fraction for Endurance, All-Electric
WF_landing=0.995+0.0025; % Weight Fraction for Landing, 1/2 for Electric, Raymer, Section 3.4
WF_fuel=1.06*(1-WF_TO*WF_climb*WF_cruise*WF_endure*WF_cruise*WF_landing); % Weight
Fraction for Fuel, Raymer, Section 3.4
ICE_m=ICE_P_W/1000; % Engine Mass of HEUAV (kg), Assume 1.5 lb/hp or 1 kg/kW
ICE_W=ICE_m*g; % ICE Weight (N)
fuel_m=WF_fuel*Wo/g; % Fuel mass of HEUAV (kg)
fuel_W=fuel_m*g; % Fuel Weight (N)
pay_m=des_uav_m-uav_glider_m-ICE_m-fuel_m-clutch_m-bat_m-EM_m-prop_m; % Payload Mass of
HEUAV (kg)
pay_W=pay_m*g; % Payload Weight of HEUAV (N)
WF_pay=pay_W/Wo; % Weight Fraction for the Payload
uav_empty_m=des_uav_m-fuel_m-pay_m; % Empty Mass of HEUAV (kg)
propulsion_m=ICE_m+clutch_m+bat_m+EM_m+prop_m; % Mass of Propulsion System for HEUAV (kg)
WF_propulsion=propulsion_m*g/Wo; % Weight Fraction for Propulsion System for Original UAV

% Check Max Speed Requirement
disp(' ')
if (ICE_P_W+EM_overtrq*EM_P_W)>(uav_PR_Vmax/prop_eff) % Over-Torque EM for short periods
   disp(['HE Propulsion System Meets Vmax Requirement by (W):  ',
num2str((ICE_P_W+EM_overtrq*EM_P_W)-(uav_PR_Vmax/prop_eff))]);
else
   disp(['HE Propulsion System Does not Meet Vmax Requirement by (W):  ',
num2str((uav_PR_Vmax/prop_eff)-(ICE_P_W+EM_overtrq*EM_P_W))]);
end

% Check Take-off Requirement
if (ICE_P_W)>(uav_PR_TO/prop_eff)
   disp(['HE ICE Alone Meets Take-off Requirement by (W):  ', num2str(ICE_P_W-
(uav_PR_TO/prop_eff))]);
else
   disp(['HE ICE Alone Does Not Meet Take-off Requirement by (W):  ', num2str((uav_PR_TO/prop_eff)-
ICE_P_W)]);
end

% Check the ROC Requirement Requirement
ROCmax=(ICE_P_W*prop_eff-uav_PR_Vendure)/uav_W; % Max ROC (m/s), Anderson-P&D, Eqns
5.117 and 5.118
if (ROCmax>perf_ROC)
   disp(['HEUAV ICE Alone Exceeds ROC Requirement by (ft/min):  ', num2str(196.9*(ROCmax-
perf_ROC))]);
```

```
else
   disp(['HEUAV ICE Alone Does Not Meet ROC Requirement by (ft/min):  ', num2str(196.9*(perf_ROC-
ROCmax))]);
end

% Check the Service Ceiling Requirement
% Best ROC at Endurance Speed, ROC=(PA-PRendure)/W
% Reference:  Anderson-P&D, Eqn 5.117, 118

% Plot Normalized Weight Fractions for UAV
figure; colormap('gray');
h=bar([uav_glider_m/des_uav_m uav_glider_m/des_uav_m; propulsion_m_org/des_uav_m
propulsion_m/des_uav_m; uav_empty_m_org/des_uav_m uav_empty_m/des_uav_m;
fuel_m_org/des_uav_m fuel_m/des_uav_m; pay_m_org/des_uav_m pay_m/des_uav_m],'group');
set(gca,'XTickLabel',{'Glider';'Propulsion';'Empty';'Fuel';'Payload'});
xlabel('Component'); ylabel('Weight Fraction');
legend('Original (ICE Only)','Hybrid-Electric');
title('Weight Fractions, Normalized to UAV Weight (m=13.6 kg, 30 lbs)');
%print -depsc2 -tiff -r600 UAV_WeightFractionsUAV

% Plot Normalized Fuel Weight Fractions for UAV Configurations
figure; colormap('gray');
bar([1-WF_TO_org 1-WF_TO; 1-WF_climb_org 1-WF_climb; 1-WF_cruise_org^2 1-WF_cruise^2; 1-
WF_endure_org 1-WF_endure;1-WF_landing_org 1-WF_landing],'group')
set(gca,'XTickLabel',{'Take-off';'Climb';'Cruise';'Endurance';'Landing'});
xlabel('Mission Segment'); ylabel('Weight Fraction');
legend('Original (ICE Only)','Hybrid-Electric');
title('Fuel Weight Fractions, Normalized to UAV Weight (m=13.6 kg, 30 lbs)');
%print -depsc2 -tiff -r600 UAV_FuelWeightFractionsUAV

% Plot Normalized Weight Fractions for the Propulsion Components for Original Configuration
figure;
bar([fuel_m_org/des_uav_m; ICE_m_org/des_uav_m; prop_m/des_uav_m],'k');
set(gca,'XTickLabel',{'Fuel';'ICE';'Prop'});
xlabel('Propulsion System Component'); ylabel('Weight Fraction');
title('Weight Fractions for Original UAV (m=13.6 kg, 30 lbs)');
%print -depsc2 -tiff -r600 UAV_WeightPropFractionsOrg

% Plot Normalized Weight Fractions for the Propulsion Components for HEUAV
figure;
bar([fuel_W/uav_W; ICE_W/uav_W; clutch_W/uav_W; bat_W/uav_W; EM_W/uav_W;
prop_W/uav_W],'k');
set(gca,'XTickLabel',{'Fuel';'ICE';'Clutch';'Batteries';'EM';'Propeller'});
xlabel('Propulsion System Component'); ylabel('Weight Fraction');
title('Weight Fractions for Hybrid-Electric UAV (m=13.6 kg, 30 lbs)');
%print -depsc2 -tiff -r600 UAV_WeightPropFractionsHE

% Display Data
disp(' ');
disp('UAV Parameters:');
disp(['UAV Oswald Efficiency Factor:  ', num2str(uav_e)]);
disp(['UAV Zero-Lift Drag Coefficient (Cdo):  ', num2str(uav_Cdo)]);
disp(['Max Lift Coefficient (Clmax):  ', num2str(uav_Clmax)]);
disp(['Max Take-off Lift Coefficient (ClmaxTO):  ', num2str(uav_ClmaxTO)]);
disp(['Propeller Efficiency:  ', num2str(prop_eff)]);
disp(' ');
disp('Performance Parameters:');
disp(['Endurance Time (hr): ', num2str(perf_tendure/3600)]);
disp(['Rate-of-Climb (ft/min): ', num2str(perf_ROC*60*3.28)]);
disp(['Cruise Velocity (kts): ', num2str(msec2kts*perf_Vcruise)]);
disp(['Endurance Velocity-Desired (kts): ', num2str(msec2kts*perf_Vendure)]);
disp(['Endurance Velocity-Actual (kts): ',num2str(msec2kts*Vendure)]);
disp(['Endurance Parameter-Actual (Cl^1.5/Cd): ', num2str(uav_Cl_15_Cd)]);
disp(['Stall Velocity-Desired (kts): ', num2str(msec2kts*perf_Vstall)]);
disp(['Stall Velocity-Actual (kts): ',num2str(msec2kts*Vstall)]);
disp(['Max Velocity (kts): ', num2str(msec2kts*perf_Vmax)]);
disp(['Take-off Distance, Ground Roll only (m): ', num2str(perf_s_TO)]);
disp(['Payload Power (W): ', num2str(des_pay_P)]);
disp(' ');
disp('UAV Design Results:');
disp(['UAV Total Mass-Desired (kg): ', num2str(des_uav_m)]);
disp(['UAV Total Mass-Actual (kg): ', num2str(Wo/g)]);
disp(['Payload Mass-Desired (kg): ', num2str(des_pay_m)]);
disp(['Payload Mass-Actual for Original (kg): ', num2str(pay_m_org)]);
disp(['Payload Mass-Actual for HEUAV (kg): ', num2str(pay_m)]);
disp(['Original UAV Empty Mass (kg): ', num2str(uav_empty_m_org)]);
disp(['HEUAV Empty Mass (kg): ', num2str(uav_empty_m)]);
disp(['Wing Area-Desired (m^2): ', num2str(des_uav_S)]);
disp(['Wing Area-Actual (m^2): ', num2str(uav_S)]);
disp(['Aspect Ratio-Desired: ', num2str(des_uav_AR)]);
disp(['Aspect Ratio-Actual: ', num2str(uav_AR)]);
disp(['Max L/D Ratio: ', num2str(LDmax)]);
disp(['L/D at Vcruise: ', num2str(LDcruise)]);
disp(['Velocity at Max L/D Ratio (kts): ', num2str(msec2kts*V_LDmax)]);
disp(['Best Glide Ratio, 1/LDmax (deg: ', num2str(180/(LDmax*pi))]);
disp(['Rate of Descent at Best Glide Ratio (ft/min): ', num2str(60*3.28*V_LDmax*sin(1/LDmax))]);
disp(['Design Lift Coefficient for Endurance: ', num2str(design_Cl_endure)]);
disp(['Design Lift Coefficient for Cruise: ', num2str(design_Cl_cruise)]);
disp(['Wing Span (m): ', num2str(uav_wing_span)]);
disp(['Wing Chord (m): ', num2str(uav_wing_chord)]);
disp(['Wing Reynolds Number, Endurance: ', num2str(uav_wing_Reynolds_Vendure)]);
disp(['Wing Reynolds Number, Cruise: ', num2str(uav_wing_Reynolds_Vcruise)]);
disp(['Wing Reynolds Number, Max Velocity: ', num2str(uav_wing_Reynolds_Vmax)]);
disp(' ');
disp('Power Requirements for the UAV:');
disp(['Power Required for Take-off (W): ', num2str(uav_PR_TO)]);
disp(['Power Required for Climb (W): ', num2str(uav_PRmin_climb)]);
disp(['Power Required for Cruise (W): ', num2str(uav_PR_Vcruise)]);
disp(['Power Required for Endurance (W): ', num2str(uav_PR_Vendure)]);
disp(['Power Required for Max Velocity (W): ', num2str(uav_PR_Vmax)]);
disp(' ');
disp('Weight Fractions for Original Configuration:');
disp(['WF for Take-off: ', num2str(WF_TO_org)]);
disp(['WF for Climb: ', num2str(WF_climb_org)]);
disp(['WF for Cruise: ', num2str(WF_cruise_org)]);
disp(['WF for Endure: ', num2str(WF_endure_org)]);
disp(['WF for Landing: ', num2str(WF_landing_org)]);
disp(['WF-Empty: ', num2str(WF_empty)]);
disp(['WF-Fuel: ', num2str(WF_fuel_org)]);
disp(['WF-Payload: ', num2str(WF_pay_org)]);
disp(['WF-Propulsion (ICE, Gen, Prop): ', num2str(WF_propulsion_org)]);
disp(['ICE Size (W): ', num2str(ICE_P_W_org)]);
disp(['ICE Size (hp): ', num2str(ICE_P_hp_org)]);
disp(['ICE Size (in^3): ', num2str(ICE_Size_in3_org)]);
disp(' ');
disp('Weight Fractions for HEUAV:');
disp(['WF for Take-off: ', num2str(WF_TO)]);
disp(['WF for Climb: ', num2str(WF_climb)]);
disp(['WF for Cruise: ', num2str(WF_cruise)]);
disp(['WF for Endure: ', num2str(WF_endure)]);
disp(['WF for Landing: ', num2str(WF_landing)]);
disp(['WF-Empty: ', num2str(WF_empty)]);
disp(['WF-Fuel: ', num2str(WF_fuel)]);
disp(['WF-Payload: ', num2str(WF_pay)]);
disp(['WF-Propulsion (ICE, Clutch, Batteries, EM, Prop): ', num2str(WF_propulsion)]);
disp(' ');
disp('Propulsion Requirements for HEUAV:');
disp(['ICE Size (W): ', num2str(ICE_P_W)]);
disp(['ICE Size (hp): ', num2str(ICE_P_hp)]);
disp(['ICE Size (in^3): ', num2str(ICE_Size_in3)]);
disp(['Power Required for Endurance (W): ', num2str(uav_PR_Vendure)]);
if x(5)<x(4)+sdmin
   V=x(4)+sdmin; % Endurance Velocity (m/s), 3 kt margin above stall
   Cl=uav_WL./(0.5*rho*V.^2); % Lift Coefficient, Anderson-Flight, Eqn 6.26
   Cd=uav_Cdo+uav_K*Cl.^2; % Drag Coefficient, Anderson-Flight, pg 359
   PR_SL=sqrt(2*uav_W^3*Cd.^2./(rho*uav_S*Cl.^3)); % PR for S&L (W), Anderson-Flight, Eqn 6.27
   disp(['Vstall>Vendure, Additional Power Required for Increased Endurance Velocity (W): ',
num2str(PR_SL-uav_PR_Vendure)]);
end
disp(['EM Power for Endurance (W): ', num2str(EM_P_W)]);
disp(['EM Efficiency (%): ', num2str(100*EM_eff)]);
disp(['EM Over-Torque Factor: ', num2str(EM_overtrq)]);
disp(['Battery Mass (kg): ', num2str(bat_m)]);
disp(['Battery Storage (Wh): ', num2str(bat_m*bat_ED)]);


function
f=SizeEqMinPR(x,uav_W,uav_Cdo_ref,uav_S_ref,uav_e,prop_eff,perf_Vstall,perf_Vendure,perf_Vcruise,
perf_Vmax,rho,sdmin)

% Nonlinear Equation to Minimize (Power Required for Endurance)

% References
% Anderson, Intro to Flight, 4th Edition
% Anderson, Aircraft Performance and Design
% Raymer, Aircraft Design:  A Conceptual Approach

% Variables
% x(1):  Wing Loading, W/S (N/m^2)
% x(2):  Aspect Ratio, AR
% x(3):  Clmax
% x(4):  Vstall (m/s)
% x(5):  Vendure (m/s)
% x(6):  Internal Combustion Engine Power (W)
% x(7):  Zero-Lift Drag Coefficient

% Function to Minimize (Power Required for Endurance)(Anderson, P&D, Eqns 5.38 & 5.56)
f=(uav_W*sqrt(2*x(1)/rho)*4*x(7)/(3*x(7)*pi*uav_e*x(2))^0.75)/prop_eff;
```

```
function
[c,ceq]=SizeEqCon(x,uav_W,uav_Cdo_ref,uav_S_ref,uav_e,prop_eff,perf_Vstall,perf_Vendure,perf_Vcrui
se,perf_Vmax,rho,sdmin)
% Constraints for Minimized Nonlinear Equation

% References
% Anderson, Intro to Flight, 4th Edition
% Anderson, Aircraft Performance and Design
% Raymer, Aircraft Design:  A Conceptual Approach

% Variables
% x(1):  Wing Loading, W/S (N/m^2)
% x(2):  Aspect Ratio, AR
% x(3):  Clmax
% x(4):  Vstall (m/s)
% x(5):  Vendure (m/s)
% x(6):  Internal Combustion Engine Power (W)
% x(7):  Zero-Lift Drag Coefficient

% Nonequality Constraint
c=[x(4)-sdmin-x(5)];

% Equality Constraints
ceq=[2*x(1)/(rho*x(3))-x(4)^2;
   2*x(1)/(rho*sqrt(3*x(7)*pi*uav_e)*x(5)^2)-sqrt(x(2));
   0.5*rho*perf_Vcruise^3*uav_W*x(7)/x(1)+2*uav_W*x(1)/(rho*perf_Vcruise*pi*uav_e*x(2))-
prop_eff*0.8*x(6);
   uav_Cdo_ref*uav_S_ref*x(1)*(10/x(5))^0.2/uav_W-x(7)];
```

A.2 HEUAV Model MATLAB Code

The HEUAV model was developed in a MATLAB/Simulink environment. Data files in the form of m-files were used to store the data for the propulsion system components and the small UAV. A setup file allows the user to select the desired configuration, engine, controller, charging strategy, and flight profile. The m-files are included in this section.

A.2.1 HEUAV Setup File

```
% Hybrid-Electric Unmanned Aerial Vehicle Input File
close all; % Close figures
clear all; % Clear variables
pack; % Clean up memory

%%%%%%%%%% PHYSICAL CONSTANTS %%%%%%%%%%
gravity_acceleration=9.81; % (m/s^2)
air_viscosity=2E-5; % Air Viscosity (N*s/m^2), FE Text-pg 11-2
air_temp_SL=288.15; % Air Temperature at SL (K)
```

```
 air_temp=air_temp_SL; % Initialize Air Temperature variable
 air_pressure_SL=1.0133E5; % Air Pressure at SL (N/m^2)
 air_density_SL=1.225; % Sea level Air Density (kg/m^3)
 air_density=air_density_SL; % Initialize Air Density variable (kg/m^3)
 load std_atm; % Load Standard Atmosphere data
 %%%%%%%%%% END OF PHYSICAL CONSTANTS %%%%%%%%%%

 %%%%%%%%%% CONFIGURATION DATA %%%%%%%%%%
 % Select configuration
 disp(' ');
 disp('Select original or hybrid-electric configuration:');
 disp('  1:  Original (engine only)');
 disp('  2:  Hybrid-Electric');
 config=input('Enter your selection:  ');
 switch config
   case 1
     HEUAV=0;
     ann=0;
     input_size=1;
     quant(1)=1;
     quant(2)=1;
     quant(3)=1;
     quant(4)=1;
     cmac_output=1;
     spd_min=1;
     trq_min=1;
     SOC_min=1;
     spd_max=1;
     trq_max=1;
     SOC_max=1;
     spd_res=1;
     trq_res=1;
     SOC_res=1;
     u(1)=0;
     u(2)=0;
     u(3)=0;
   case 2
     % Select charging strategy
     disp(' ');
     disp('Select a charging strategy:');
     disp('  1:  Charge-Sustaining');
     disp('  2:  Charge-Depletion');
     chg_strategy=input('Enter your selection:  ');
     HEUAV=1;
     ann=1;
     u(1)=0;
     u(2)=0;
     u(3)=0;
   case 3
     disp('Not a valid selection.');
end

% Select type of engine
disp(' ');
disp('Select a specific engine:');
disp('  1:  FCQ100 or PC125->2-Stroke Gasoline Engine, Scaled');
disp('  2:  GX31->4-Stroke Gasoline Engine');
fc_sel=input('Enter your selection:  ');
switch fc_sel
   case 1
      FC_Q100; % Quadra Aerrow 100 cm^3 2-Stroke Engine, from Ferguson
      %FC_PC125; % Piaggio Cosa 125 cm^3 2-Stroke Engine, from Heywood and Sher
   case 2
      FC_HondaGX31; % Honda GX31 4-Stroke Gasoline Engine
   case 3
      disp('Not a valid selection.');
end

% If HEUAV configuration, select type of controller
NN_Strategy=0;
if (HEUAV==1)
   disp(' ');
   disp('Select Rule-Based or CMAC ANN controller:');
   disp('  1:  Rule-Based');
   disp('  2:  CMAC ANN');
   controller=input('Enter your selection:  ');
   if (controller==2)
      NN_Strategy=1;
   end
   switch controller
      case 1
         input_size=1;
         quant(1)=1;
         quant(2)=1;
         quant(3)=1;
         quant(4)=1;
         cmac_output=1;
         spd_min=1;
         trq_min=1;
         SOC_min=1;
         spd_max=1;
         trq_max=1;
         SOC_max=1;
         spd_res=1;
         trq_res=1;
         SOC_res=1;
      case 2
         disp(' ');
         disp('Select the CMAC ANN controller:');
         disp('  1:  2 Inputs-Rotational Speed and Demanded Torque');
         disp('  2:  3 Inputs-Rotational Speed, Demanded Torque, and SOC');
         ann=1+input('Enter your selection:  ');
         switch ann
            case 2
               input_size=2;
               quant(1)=70;
               quant(2)=51;
               quant(3)=1;
               if (fc_sel==1)
                  fid=fopen('Trq_Cmd_ICE_EM_2S_MATLAB_data.txt');
               else
                  fid=fopen('Trq_Cmd_ICE_EM_4S_MATLAB_data.txt');
      end
      [A,count]=fscanf(fid,'%15f'); % 8f for CMAC code, 15f if using optimization data directly
      c=1;
      for (i=1:quant(1))
         for (j=1:quant(2))
            input1(i)=A(c);
            input2(j)=A(c+1);
            cmac_output(i,j)=A(c+2);
            c=c+3;
         end
      end
      spd_min=input1(1);
      trq_min=input2(1);
      SOC_min=1;
      spd_max=input1(quant(1));
      trq_max=input2(quant(2));
      SOC_max=1;
      spd_res=input1(2)-input1(1);
      trq_res=input2(2)-input2(1);
      SOC_res=1;
   case 3
      input_size=3;
      quant(1)=70;
      quant(2)=51;
      quant(3)=50;
      if (fc_sel==1)
         if (chg_strategy==1)
            fid=fopen('Trq_Cmd_ICE_EM_SOC_2S_CS_MATLAB_data.txt');
         else
            fid=fopen('Trq_Cmd_ICE_EM_SOC_2S_CD_MATLAB_data.txt');
         end
      else
         if (chg_strategy==1)
            fid=fopen('Trq_Cmd_ICE_EM_SOC_4S_CS_MATLAB_data.txt');
         else
            fid=fopen('Trq_Cmd_ICE_EM_SOC_4S_CD_MATLAB_data.txt');
         end
      end
      [A,count]=fscanf(fid,'%15f'); % 8f for CMAC code, 15f if using optimization data directly
      c=1;
      for (i=1:quant(1))
         for (j=1:quant(2))
            for (k=1:quant(3))
               input1(i)=A(c);
               input2(j)=A(c+1);
               input3(k)=A(c+2);
               cmac_output(i,j,k)=A(c+3);
               c=c+4;
            end
         end
      end
      spd_min=input1(1);
      trq_min=input2(1);
      SOC_min=input3(1);
      spd_max=input1(quant(1));
      trq_max=input2(quant(2));
```

```
            SOC_max=input3(quant(3));
            spd_res=input1(2)-input1(1);
            trq_res=input2(2)-input2(1);
            SOC_res=input3(2)-input3(1);
          case 4
            disp('Not a valid selection.');
        end
      case 3
        disp('Not a valid selection.');
    end
    spd_index=1;
    trq_index=1;
end
%%%%%%%%%% END OF CONFIGURATION DATA %%%%%%%%%%

%%%%%%%%%% SPEED/ALTITUDE PROFILE %%%%%%%%%%
% Load speed/altitude profile
disp(' ');
disp('Choose a speed/altitude profile:');
disp('  1:  Endurance Speed-25 kts, 5 kft MSL');
disp('  2:  Cruise Speed-50 kts, 5 kft MSL');
disp('  3:  Max Speed-65 kts, 5 kft MSL');
disp('  4:  Regen-35 kts, descent');
disp('  5:  Combo-climb, cruise, endurance, descent, etc.');
disp('  6:  ISR Mission-1 hr cruise, 1 hr endurance, 1 hr cruise');
disp('  7:  Georgia ISR Mission-includes 2 endurance segments');
disp('  8:  Terrain Following-50 kts, 5 kft MSL, +/-100 ft');
profile=input('Enter your selection:  ');
disp(' ');

switch profile
  case 1
    PR_CON25KTS;
  case 2
    PR_CON50KTS;
  case 3
    PR_MAXSPD;
  case 4
    PR_REGEN;
  case 5
    PR_COMBO;
    pr_filter_bool=1;
    pr_avg_time=2.5;
  case 6
    PR_ISR;
    pr_filter_bool=1;
    pr_avg_time=2.5;
  case 7
    PR_ISR_GA;
    pr_filter_bool=1;
    pr_avg_time=2.5;
  case 8
    PR_TF;
    pr_filter_bool=1;
    pr_avg_time=2.5;
  otherwise
```

```
        PR_COMBO;
        pr_filter_bool=1;
        pr_avg_time=2.5;
    end

    % Select if a wind gust disturbance is desired
    disp('Do you want to include wind turbulence?: ');
    disp(' 0: No');
    disp(' 1: Yes');
    wg_on=input('Enter your selection: ');
    disp(' ');

    % Start and stop times
    sim_start=max(pr_kts(min(find(pr_kts(:,2)>0)),1)-2,0); % (s)
    sim_stop=max(pr_kts(:,1)); % (s)
    %%%%%%%%%% END OF SPEED/ALTITUDE PROFILE %%%%%%%%%%

    %%%%%%%%%% UAV & COMPONENT DATA %%%%%%%%%%
    % Load component and UAV data
    MC_Aveox2739_3Y; % Aveox 2739/3Y motor, sized for endurance speed
    ESS_UBI2590_Parallel; % Ultralife UBI-2590 Battery, sized for endurance speed
    UAV_30lb; % Includes drag polar and prop data
    %%%%%%%%%% END OF UAV & COMPONENT DATA %%%%%%%%%%

    %%%%%%%%%% MISCELLANEOUS DATA %%%%%%%%%%
    radps2rpm=60/(2*pi); % (rad/s->rpm)
    workspace_sample_time=-1; % Sample time of -1 saves variables at simulation step size
    %%%%%%%%%% END OF MISCELLANEOUS DATA %%%%%%%%%%

    %%%%%%%%%% PILOT MODEL DATA %%%%%%%%%%
    % PID Controller
    % For ISR_GA, can use HEUAV values for the original configuration
    if(config==1) % Original configuration
      pil_accel_P=0.25;
      pil_accel_I=0.1;
      pil_accel_D=0.1;
      pil_decel_P=0.25;
      pil_decel_I=0.1;
      pil_decel_D=0.1;
      pil_I_clamp=0.5;
    else % HEUAV configuration
      pil_accel_P=0.1; % 2S-0.1, 4S-0.5
      pil_accel_I=0.02; % 2S-0.02, 4S-0.1
      pil_accel_D=0.02; % 2S-0.02, 4S-0.1
      pil_decel_P=0.1; % 2S-0.1, 4S-0.5
      pil_decel_I=0.02; % 2S-0.02, 4S-0.1
      pil_decel_D=0.02; % 2S-0.02, 4S-0.1
      pil_I_clamp=0.5; %
    end
    % First-order transfer function
    pil_tau=0.05; % (s) Original: 0.05
    % Second-order transfer function
    zeta=1; % Damping Ratio
    wn=25; % Natural Frequency (rad/s)
    pil_tf_a=2*zeta*wn;
    pil_tf_b=wn^2;
```

```
clear zeta, wn;
% Other settings
pil_max_rate=10; % Throttle throw rate of change limit (throttle throw/s)
pil_accel_max=0.999; % Maximum Throttle Throw (1.0=full command, 0.999 required to prevent
numerical problems)
pil_decel_max=0.999; % Maximum Regenerative Throw (1.0=full command)
%%%%%%%%%% END OF PILOT MODEL DATA %%%%%%%%%%

%%%%%%%%% ENGINE DATA %%%%%%%%%
% Determine ICE inertia
if (HEUAV==0)
  fc_inertia=eng_mass_org*(0.5*0.5*0.05^2); % Engine Inertia (kg*m^2), Original engine
  fc_inertia_org=fc_inertia;
else
  fc_inertia=fc_mass*(0.5*0.5*0.05^2); % Engine Inertia (kg*m^2), Down-sized engine for HEUAV
  fc_inertia_org=fc_inertia;
end
% Assumes 50% of engine is equivalent to rotating parts and the rotating mass is a cylinder
% The equivalent cylinder mass has a radius of 0.05 m
% Optional Reference:  Auto. Handbook, Bosch, 1996, pg 385->need piston and crankshaft mass
%%%%%%%%% END OF ENGINE DATA %%%%%%%%%

%%%%%%%%%%% CLUTCH DATA %%%%%%%%%%%
% Clutch torque response transfer function coefficients
clutch_inertia=3.5E-5; % Clutch Inertia (kg*m^2), Reference:  RM Hoffman Company
zeta=1; % Damping Ratio
wn=25; % Natural Frequency (rad/s)
clutch_tf_a=2*zeta*wn;
clutch_tf_b=wn^2;
clutch_max_torque=2; % Max Torque for Clutch (N*m)
clutch_mus_over_muk=1.2;
clutch_engagement_time=1.0; % Engagement Time (s)
clutch_disengagement_time=1.0; % Disengagement Time (s)
clear zeta, wn;
%%%%%%%%%%% END OF CLUTCH DATA %%%%%%%%%%%

%%%%%%%%% BATTERY DATA %%%%%%%%%
% No additional data
%%%%%%%%% END OF BATTERY DATA %%%%%%%%%

%%%%%%%%% MOTOR DATA %%%%%%%%%
% No additional data
%%%%%%%%% END OF MOTOR DATA %%%%%%%%%

%%%%%%%%% PROP DATA %%%%%%%%%
% No additional data
%%%%%%%%% END OF PROP DATA %%%%%%%%%

%%%%%%%%% UAV DATA %%%%%%%%%
% No additional data
%%%%%%%%% END OF UAV DATA %%%%%%%%%

%%%%%%%%% POWERTRAIN CONTROLLER DATA %%%%%%%%%
ptc_hybrid_engine_on_spd_low=13.5; % UAV Speed above max endurance speed (m/s)
ptc_hybrid_engine_on_spd_high=14.5; % UAV Speed above max endurance speed (m/s)
ptc_SOC_low1=0.15; % Lowest SOC before using a CS algorithm
```

```
ptc_SOC_low2=0.20; % SOC level to turn off CS, permits hysteresis during CS operation
ptc_CS_P=0.5; % ISR_GA_4S_CS, ISR_GA_2S_CS, or Combo_4S-0.5, Cruise_Wind_4S-10
ptc_CS_D=0.1;
mc_IOL_trq=mc_max_trq;
enable_stop=0; % Enable Stop: 0: Do not Enable, 1: Enable Stop (0.001<SOC<100, ESS Voltage out of
range, or Run out of Fuel)
zeta=1; % Damping Ratio
wn=25; % Natural Frequency (rad/s)
cmac_tf_a=2*zeta*wn;
cmac_tf_b=wn^2;
clear zeta, wn;
%%%%%%%% END OF POWERTRAIN CONTROLLER DATA %%%%%%%%

%%%%%%%% DISPLAY PERTINENT DATA %%%%%%%%
disp(' ');
disp('Pertinent data for the simulation:');
disp(['Configuration (0: Original, 1: HEUAV) - ', num2str(HEUAV)]);
disp(['Control Strategy (0: Rule-Based, 1: NN) - ', num2str(NN_Strategy)]);
disp(' ');
disp(['Note: Mass data only applicable if HEUAV Configuration selected']);
disp(['UAV Total Mass (kg) - ', num2str(uav_total_mass)]);
disp(['UAV Glider Mass (kg) - ', num2str(uav_glider_mass)]);
disp(['Original UAV Payload Mass (kg) - ', num2str(uav_payload_mass_org)]);
disp(['HEUAV Payload Mass (kg) - ', num2str(uav_payload_mass)]);
disp(['Original UAV Empty Mass (kg) - ', num2str(uav_empty_mass_org)]);
disp(['HEUAV Empty Mass (kg) - ', num2str(uav_empty_mass)]);
disp(['Battery Mass (kg) - ', num2str(ess_module_mass*ess_module_num)]);
disp(' ');
disp(['Battery Capacity (Ah) - ', num2str(ess_max_ah_capacity)]);
disp(['Battery Maximum Volts (V) - ', num2str(ess_module_num*ess_max_volts)]);
disp(['Battery Storage (Wh) - ', num2str(ess_max_ah_capacity*ess_module_num*ess_max_volts)]);
disp(['Engine Torque Scale - ', num2str(fc_trq_scale)]);
disp(['Motor Torque Scale - ', num2str(mc_trq_scale)]);
fclose('all');
%%%%%%%% END OF DISPLAY PERTINENT DATA %%%%%%%%
```

A.2.2 Internal Combustion Engine Data Files

```
% Engine Data File: FC_Q100.M

% Data sources:
% Quadra Aerrow Engines, (613)264-0010, Perth, Ontario
% ICE, Ferguson, 1986, p. 480, 356 cm^3 engine
% Other references as mentioned in the comments

% Q100B Gasoline Engine (Battery Ignition)
% Maximum Power: 7.8 kW (10.4 hp) at 7800 rpm
% Peak Torque: 9.6 N*m (7.1 ft-lbs) at 7600 rpm

%%%%%%%%% FILE ID INFO %%%%%%%%%
fc_description='Quadra Aerrow Gasoline Engine, 7.8 kW (10.4 hp)';
fc_proprietary=0; % 0=> non-proprietary, 1=> proprietary
fc_validation=1; % 0=> no validation, 1=> data agrees with source data
fc_fuel_type='Gasoline';
fc_disp=0.098; % Engine Displacement (L)
```

```
disp(['Data loaded:  FC_Q100.M - ',fc_description]);
%%%%%%%%%% END OF FILE ID INFO %%%%%%%%%%

%%%%%%%%%% DEFAULT SCALING %%%%%%%%%%
% Scale fc_map_spd to simulate a faster or slower running engine (--)
fc_spd_scale=0.85;
% Scale fc_map_trq to simulate a higher or lower torque engine (--)
if (HEUAV==0)
   fc_trq_scale=1.5*0.21; % Original engine
else
   fc_trq_scale=1.0*0.21; % Down-sized engine for HEUAV
end
% Scale fc power (--)
fc_pwr_scale=fc_spd_scale*fc_trq_scale;
%%%%%%%%%% END OF DEFAULT SCALING %%%%%%%%%%

%%%%%%%%%% SPEED & TORQUE RANGES %%%%%%%%%%
% Speed range of the engine (rad/s)
fc_map_spd=fc_spd_scale*[2000 3000 4000 5000 6000 7000 8000 9000 10000]*(2*pi)/60;
% Torque range of the engine (N*m)
fc_map_trq=fc_trq_scale*[1.5 2.5 3.5 4.5 5.5 6.5 7.5]*1.3558;
%%%%%%%%%% END OF SPEED & TORQUE RANGES %%%%%%%%%%

%%%%%%%%%% FUEL USE MAPS %%%%%%%%%%
% Fuel use map indexed vertically by fc_map_spd (top to bottom)
% and horizontally by fc_map_trq (left to right) (g/kWh)
% Efficiency map estimated from ICE, Ferguson, 1986, p. 480
fc_fuel_map_gpkWh=1.5*[1300 1140 1120 1130 1180 1310 1400;
                        1280 1110 1030 1050 1100 1215 1280;
                        1200  980  870  910  930 1070 1200;
                        1225  950  785  780  785  990 1100;
                        1130  870  765  725  750  870 1020;
                        1060  920  710  600  625  780  980;
                         950  825  650  525  610  710  980;
                        1000  910  750  700  770  820  970;
                        1080  980  955  960  970 1000 1070];
% Convert g/kWh to efficiency to get fuel efficiency map
% (reciprocal of g/kWh*33.44 kWh/gal*gal/6.15 lbs*2.2 lbs/kg*kg/1000 g)
% Reference:  SAE J1711 Standards Paper
fc_eff_map=(6.15*1000)./(fc_fuel_map_gpkWh*33.44*2.2);
% Convert g/kWh maps to g/s maps
[T,w]=meshgrid(fc_map_trq, fc_map_spd);
fc_map_kW=T.*w/1000;
fc_fuel_map_gs=fc_fuel_map_gpkWh.*fc_map_kW/3600;
%%%%%%%%%% END OF FUEL USE MAPS %%%%%%%%%%

%%%%%%%%%% LIMITS %%%%%%%%%%
% Maximum torque curve of the engine indexed by fc_map_spd (N*m)
fc_max_trq=fc_trq_scale*[4.5 5.5 6.0 6.5 6.7 6.9 6.9 6.0 4.6]*1.3558;
fc_stall_spd=fc_spd_scale*1000*(2*pi)/60; % Engine Stall Speed (rad/s)
fc_rev_limit=max(fc_map_spd)*1.5; % Maximum Engine Speed (rad/s)
%%%%%%%%%% END OF LIMITS %%%%%%%%%%

%%%%%%%%%% PARAMETERS THAT SCALE %%%%%%%%%%
fc_max_pwr=(max(fc_map_spd.*fc_max_trq)/1000); % Peak Engine Power (kW)
%fc_base_mass=3.1*fc_pwr_scale; % Engine Mass (kg)
```

```
%fc_fuel_mass=2.34*fc_pwr_scale; % Mass of Fuel (kg), (0.6 kg/kWh*7.8 kW for 0.5 h)
%fc_mass=fc_base_mass+fc_fuel_mass; % Total Engine/Fuel System Mass (kg)
%fc_inertia=fc_pwr_scale*fc_base_mass*(0.5*0.5*0.05^2); % Engine Inertia (kg*m^2)
% Assumes 50% of engine is equivalent to rotating parts and the rotating mass is a cylinder
% The equivalent cylinder mass has a radius of 0.05 m
% Optional Reference: Auto. Handbook, Bosch, 1996, p.385->need piston and crankshaft mass
%%%%%%%%%% END OF PARAMETERS THAT SCALE %%%%%%%%%%

%%%%%%%%%% OTHER PARAMETERS %%%%%%%%%%
% Density and LHV of gasoline
% Reference: Thermodynamics, Cengel, 2002, pg. 864
fc_fuel_den=753; % Density of the Fuel (g/liter)
fc_fuel_lhv=44000; % Lower Heating Value of the Fuel (J/g)
% Coefficients of engine torque response transfer function
zeta=1; % Damping Ratio
wn=16; % Natural Frequency (rad/s)
fc_tf_a=2*zeta*wn;
fc_tf_b=wn^2;
clear zeta, wn;
% Idle Control
fc_idle_spd=fc_spd_scale*2500*(2*pi)/60; % Idle Speed (rad/s)
fc_idle_control_spd=1.05*fc_idle_spd; % Idle Control Speed (rad/s)
fc_idle_loop_prop=0.1; % PI-Compensator P coefficient, Original: 0.2
fc_idle_loop_int=0.1; % PI-Compensator I coefficient, Original: 0.2
% Closed throttle torque of the engine indexed by fc_map_spd (N*m)
fc_ct_trq=fc_trq_scale*(4.448/3.281*(-fc_disp)*61.02/24*(fc_map_spd/max(fc_map_spd)).^2+
14*(fc_map_spd/max(fc_map_spd))));
% Throttle Position (deg)
% Row index in fc_trq_map_throt and fc_trq_map_ign_off, fc_map_spd is the column index
fc_map_throt=[0 10 20 30 40 50 60 70 80 90];
throt_max=max(fc_map_throt);
% Throttle Exponent and Curve
throt_exp=0.5;
fc_throt_curve=((fc_map_throt/throt_max).^throt_exp);
% Torque map as a function of throttle & speed when ignition is on (N*m)
fc_trq_map_throt=meshgrid(fc_ct_trq,fc_map_throt)+((fc_map_throt/throt_max).^throt_exp)'*(fc_max_trq
-fc_ct_trq);
% Torque map as a function of throttle & speed when ignition is off (N*m)
fc_trq_map_ign_off=fc_ct_trq(1)+(1-(fc_map_throt/throt_max).^throt_exp)'*(fc_ct_trq-fc_ct_trq(1));
% Default idle throttle position, throttle for zero torque (deg)
fc_idle_throt=interp1(fc_trq_map_throt(:,1),fc_map_throt,0);
%%%%%%%%%% END OF OTHER PARAMETERS %%%%%%%%%%

%%%%%%%%%% IOL TABLE %%%%%%%%%%
[fc_max_eff,trq_index]=max(fc_eff_map');
fc_IOL_trq=fc_map_trq(trq_index);
%%%%%%%%%% END OF IOL TABLE %%%%%%%%%%


% Engine Data File: FC_HondaGX31.M

% Data sources:
% Honda Mini 4-Stroke Engine brochure
% Efficiency map based on the Insight engine map from ADVISOR
% Other references as mentioned in the comments
```

```
%%%%%%%%%% FILE ID INFO %%%%%%%%%%
fc_description='Honda 4-Stroke Gasoline Engine, 31 cc';
fc_proprietary=0; % 0=> non-proprietary, 1=> proprietary
fc_validation=1; % 0=> no validation, 1=> data agrees with source data
fc_fuel_type='Gasoline';
fc_disp=0.031; % Engine Displacement (L)
disp(['Data loaded:  FC_HondaGX31.M - ',fc_description]);
%%%%%%%%%%% END OF FILE ID INFO %%%%%%%%%%

%%%%%%%%%%% DEFAULT SCALING %%%%%%%%%%
% Scale fc_map_spd to simulate a faster or slower running engine (--)
fc_spd_scale=1.5;
% Scale fc_map_trq to simulate a higher or lower torque engine (--)
if (HEUAV==0)
   fc_trq_scale=1.5*0.024; % Original engine
else
   fc_trq_scale=1.0*0.024; % Down-sized engine for HEUAV
end
% Scale fc power (--)
fc_pwr_scale=fc_spd_scale*fc_trq_scale;
%%%%%%%%%%% END OF DEFAULT SCALING %%%%%%%%%%

%%%%%%%%%%% SPEED & TORQUE RANGES %%%%%%%%%%
% Speed range of the engine (rad/s)
fc_map_spd=fc_spd_scale*[1200 1600 2000 3200 4000 4400 4800 5600 6000]*2*pi/60;
% Torque range of the engine (N*m)
fc_map_trq=fc_trq_scale*[6.8 13.6 20.4 27.2 33.8 40.6 47.4 54.2 61 67.8 74.6 81.4];
%%%%%%%%%%% END OF SPEED & TORQUE RANGES %%%%%%%%%%

%%%%%%%%%%% FUEL USE MAPS %%%%%%%%%%
% Fuel use map indexed vertically by fc_map_spd (top to bottom)
% and horizontally by fc_map_trq (left to right) (g/kWh)
fc_fuel_map_gpkWh=1.4*[
685.7   635.7   541.4   447.2   352.9   332.2   311.4   322.4   333.5   333.5   333.5   333.5
678.4   500.1   443.8   387.4   331.1   301.8   297     263.4   269.8   328     335     335
663.4   483.4   407.6   350.1   294.3   280.8   267.3   253.9   269.8   303.2   336.7   336.7
699.1   537.9   480.3   412.7   301.4   283.9   266.3   248.7   258.8   268.8   271.9   317.9
662.9   592.9   494.6   393.4   295.1   279.4   263.6   247.9   255.2   262.5   295     320
667.9   524.8   381.6   351.9   322.2   304.9   287.5   270.8   290.8   310.9   320.9   324
650.6   530.6   422.5   411.1   310     305     305.8   304.2   314.5   324.8   332.7   337.7
698.4   500.5   428.6   392.7   356.8   337.9   328.4   319     328.8   333.6   338.7   340.7
751.1   637.8   521.1   407.8   393.1   378.4   363.3   348.2   338.8   340.2   345.2   350.2];

% Convert g/kWh to efficiency to get fuel efficiency map
% (reciprocal of g/kWh*33.44 kWh/gal*gal/6.15 lbs*2.2 lbs/kg*kg/1000 g)
% Reference:  SAE J1711 Standards Paper
fc_eff_map=(6.15*1000)./(fc_fuel_map_gpkWh*33.44*2.2);
% Convert g/kWh maps to g/s maps
[T,w]=meshgrid(fc_map_trq, fc_map_spd);
fc_map_kW=T.*w/1000;
fc_fuel_map_gs=fc_fuel_map_gpkWh.*fc_map_kW/3600;
%%%%%%%%%%% END OF FUEL USE MAPS %%%%%%%%%%

%%%%%%%%%%% LIMITS %%%%%%%%%%
% Maximum torque curve of the engine indexed by fc_map_spd (N*m)
fc_max_trq=fc_trq_scale*[66 71 76 81 81 79 77 66 60];
```

```
fc_stall_spd=fc_spd_scale*800*(2*pi)/60; % Engine Stall Speed (rad/s)
fc_rev_limit=max(fc_map_spd)*1.5; % Maximum Engine Speed (rad/s)
%%%%%%%%%% END OF LIMITS %%%%%%%%%%

%%%%%%%%%% PARAMETERS THAT SCALE %%%%%%%%%%
fc_max_pwr=(max(fc_map_spd.*fc_max_trq)/1000); % Peak Engine Power (kW)
%fc_base_mass=3.1*fc_pwr_scale; % Engine Mass (kg)
%fc_fuel_mass=2.34*fc_pwr_scale; % Mass of Fuel (kg), (0.6 kg/kWh*7.8 kW for 0.5 h)
%fc_mass=fc_base_mass+fc_fuel_mass; % Total Engine/Fuel System Mass (kg)
%fc_inertia=fc_pwr_scale*fc_base_mass*(0.5*0.5*0.05^2); % Engine Inertia (kg*m^2)
% Assumes 50% of engine is equivalent to rotating parts and the rotating mass is a cylinder
% The equivalent cylinder mass has a radius of 0.05 m
% Optional Reference: Auto. Handbook, Bosch, 1996, p.385->need piston and crankshaft mass
%%%%%%%%%% END OF PARAMETERS THAT SCALE %%%%%%%%%%

%%%%%%%%%% OTHER PARAMETERS %%%%%%%%%%
% Density and LHV of gasoline
% Reference: Thermodynamics, Cengel, 2002, pg. 864
fc_fuel_den=753; % Density of the Fuel (g/liter)
fc_fuel_lhv=44000; % Lower Heating Value of the Fuel (J/g)
% Coefficients of engine torque response transfer function
zeta=1; % Damping Ratio
wn=25; % Natural Frequency (rad/s)
fc_tf_a=2*zeta*wn;
fc_tf_b=wn^2;
clear zeta, wn;
% Idle Control
fc_idle_spd=fc_spd_scale*1500*(2*pi)/60; % Idle Speed (rad/s)
fc_idle_control_spd=1.05*fc_idle_spd; % Idle Control Speed (rad/s)
fc_idle_loop_prop=0.5; % PI-Compensator P coefficient, Original: 0.2
fc_idle_loop_int=0.1; % PI-Compensator I coefficient, Original: 0.2
% Closed throttle torque of the engine indexed by fc_map_spd (N*m)
fc_ct_trq=fc_trq_scale*(4.448/3.281*(-fc_disp)*61.02/24*(9*(fc_map_spd/max(fc_map_spd)).^2+
14*(fc_map_spd/max(fc_map_spd))));
% Throttle Position (deg)
% Row index in fc_trq_map_throt and fc_trq_map_ign_off, fc_map_spd is the column index
fc_map_throt=[0 10 20 30 40 50 60 70 80 90];
throt_max=max(fc_map_throt);
% Throttle Exponent and Curve
throt_exp=1.0; % Original 0.5
fc_throt_curve=((fc_map_throt/throt_max).^throt_exp);
% Torque map as a function of throttle & speed when ignition is on (N*m)
fc_trq_map_throt=meshgrid(fc_ct_trq,fc_map_throt)+((fc_map_throt/throt_max).^throt_exp)'*(fc_max_trq
-fc_ct_trq);
% Torque map as a function of throttle & speed when ignition is off (N*m)
fc_trq_map_ign_off=fc_ct_trq(1)+(1-(fc_map_throt/throt_max).^throt_exp)'*(fc_ct_trq-fc_ct_trq(1));
% Default idle throttle position, throttle for zero torque (deg)
fc_idle_throt=interp1(fc_trq_map_throt(:,1),fc_map_throt,0);
%%%%%%%%%% END OF OTHER PARAMETERS %%%%%%%%%%

%%%%%%%%%% IOL TABLE %%%%%%%%%%
[fc_max_eff,trq_index]=max(fc_eff_map');
fc_IOL_trq=fc_map_trq(trq_index);
%%%%%%%%%% END OF IOL TABLE %%%%%%%%%%
```

A.2.3 Battery Data File

```
% Battery Data File:  ESS_UBI2590_Parallel.m

% Data source:  Ultralife Lithium-Ion Battery Brochure/Manufacturer Data

%%%%%%%%%%% FILE ID INFO %%%%%%%%%%%
ess_description='2 Ultralife UBI-2590 Lithium-Ion Batteries in Parallel, each in Parallel Mode';
ess_proprietary=0; % 0=> non-proprietary, 1=> proprietary, do not distribute
ess_validation=1; % 0=> no validation, 1=> data agrees with source data
disp(['Data loaded:  ESS_UBI2590_Parallel.m - ',ess_description]);
%%%%%%%%%%% END OF FILE ID INFO %%%%%%%%%%%

%%%%%%%%%%% SOC & TEMP RANGES %%%%%%%%%%%
ess_soc=[0 10 20 40 60 80 100]/100; % (--)
ess_tmp=[0 23 45]; % (C)
%%%%%%%%%%% END OF SOC & TEMP RANGES %%%%%%%%%%%

%%%%%%%%%%% LOSS & EFFICIENCY DATA %%%%%%%%%%%
% Parameters indexed by SOC horizontally and temperature vertically
% Max Capacity at C/2.5 rate, indexed by ess_tmp
ess_max_ah_cap=2*2*[4.75 5 4.9]; % (Ah)
% Coulombic Efficiency, indexed by ess_tmp
ess_coulombic_eff=[0.97 0.99 0.99]; % (--)
% Module's Resistance during Discharge, indexed by ess_soc and ess_tmp
ess_r_dis=1/4*[0.30 0.34 0.34 0.34 0.34 0.34 0.34;
               0.17 0.17 0.17 0.17 0.17 0.17 0.17;
               0.15 0.15 0.16 0.18 0.18 0.18 0.18]; % (ohm)
% Module's Resistance during Charge, indexed by ess_soc and ess_tmp
% No data available, set equal to discharge resistance
ess_r_chg=ess_r_dis;
% Module's Open-Circuit Voltage (no-load), indexed by ess_soc and ess_tmp
ess_voc=[3.57 3.63 3.69 3.77 3.86 3.97 4.05;
         3.48 3.59 3.66 3.76 3.85 3.96 4.05;
         3.40 3.51 3.59 3.75 3.83 3.96 4.05]*4; % (V)
%%%%%%%%%%% END OF LOSS & EFFICIENCY DATA %%%%%%%%%%%

%%%%%%%%%%% LIMITS %%%%%%%%%%%
ess_min_volts=12;
ess_max_volts=16.4;
%%%%%%%%%%% END OF LIMITS %%%%%%%%%%%

%%%%%%%%%%% OTHER DATA %%%%%%%%%%%
ess_module_num=1; % Default for Number of Modules
ess_module_mass=2*1.44; % Mass of 2 UBI-2590 batteries in parallel (kg)
ess_cap_scale=1; % Scale Factor for module max Ah Capacity
%%%%%%%%%%% END OF OTHER DATA %%%%%%%%%%%
```

A.2.4 Electric Motor Data File

```
% Motor Data File:  MC_Aveox2739_3Y

% Data source:
% Aveox specification sheet for the 2739/3Y brushless motor
```

```
%%%%%%%%%% FILE ID INFO %%%%%%%%%%
mc_version=1;
mc_description='Aveox, 2739/3Y Brushless Motor';
mc_proprietary=0; % 0=>non-proprietary, 1=>proprietary
mc_validation=1; % 0=>no validation, 1=>data agrees with source data
% 2=>data matches source data and data collection methods have been verified
disp(['Data loaded:  MC_Aveox2739/3Y - ',mc_description]);
%%%%%%%%%% END OF FILE ID INFO %%%%%%%%%%

%%%%%%%%%% MOTOR DATA %%%%%%%%%%
Io=0.90; % No Load Current (amps), measured with sensorless controller
Imax=30; % Maximum Continuous Current (amps)
Kv=1134; % Voltage Constant (rpm/volt), measured with sensorless controller
Kt=1.19; % Torque Constant (in-oz/amp)
Rm=0.0817; % Armature Resistance (ohms)
Km=Kt/sqrt(Rm); % Motor Constant (in-oz/sqrt(W))
Pmax=600; % Maximum Continuous Power (W)
RPMmax=50000; % Maximum Speed (rpm)
%%%%%%%%%% END OF MOTOR DATA %%%%%%%%%%

%%%%%%%%%% DEFAULT SCALING %%%%%%%%%%
% Scale mc_map_spd to simulate a faster or slower running motor (--)
mc_spd_scale=1.0;
% Scale mc_map_trq to simulate a higher or lower torque motor (--)
mc_trq_scale=1.0;
% Scale mc power (--)
mc_pwr_scale=mc_spd_scale*mc_trq_scale;
%%%%%%%%%% END OF DEFAULT SCALING %%%%%%%%%%

%%%%%%%%%% SPEED & TORQUE RANGES %%%%%%%%%%
% Calculations based on page 9 of Aveox manual
% Calculate the torque based on the current
mc_current=[1 2 3 5 7.5 10 15 20 25 30 35 40]; % Current (amps)
mc_map_trq=mc_trq_scale*(mc_current-Io)*Kt*(1/12)*(1/16)*1.3558; % Torque Range (N*m)
% Calculate the no-load speed based on the voltage
mc_voltage=[5 7.5 10 12.5 15 17.5 20 22.5 25 27.5 30]; % Voltage (volts)
mc_max_voltage=RPMmax/Kv+Rm*Io; % Maximum Voltage for no-load (V)
mc_map_spd=mc_spd_scale*(2*pi)/60*Kv*(mc_voltage-Rm*Io); % No-Load Speed Range (rad/s)
%%%%%%%%%% END OF SPEED & TORQUE RANGES %%%%%%%%%%

%%%%%%%%%% LOSSES AND EFFICIENCIES %%%%%%%%%%
% Calculate the efficiency map
[T,w]=meshgrid(mc_map_trq, mc_map_spd);
[I,w]=meshgrid(mc_current, mc_map_spd);
mc_outpwr_map2=T.*w;
mc_inpwr_map2=I.*(w*60/(Kv*2*pi)+Rm*I);
mc_eff_map1=mc_outpwr_map2./mc_inpwr_map2;

% Convert Efficiency Map to Input Power Map
% Compute losses in well-defined efficiency area
[T1,w1]=meshgrid(mc_map_trq,mc_map_spd);
mc_outpwr_map1=T1.*w1;
mc_losspwr_map1=(1./mc_eff_map1-1).*mc_outpwr_map1;

%% Compute losses in entire operating range
%% ASSUME that losses are symmetric about zero-torque axis
```

```
mc_map_trq=[-fliplr(mc_map_trq) mc_map_trq];
mc_eff_map=[fliplr(mc_eff_map1) mc_eff_map1];
mc_losspwr_map=[fliplr(mc_losspwr_map1) mc_losspwr_map1];
mc_outpwr_map=[-fliplr(mc_outpwr_map1) mc_outpwr_map1];
mc_inpwr_map=mc_outpwr_map+mc_losspwr_map;
%%%%%%%%%% END OF LOSSES AND EFFICIENCIES %%%%%%%%%%

%%%%%%%%%% LIMITS %%%%%%%%%%
mc_max_crrnt=Imax; % Maximum Current (amps)
mc_min_voltage=mc_voltage(1,1); %  Minimum Voltage (volts)
mc_max_trq1=ones(size(mc_map_spd)).*(Imax-Io)*Kt*(1/16)*(1/12)*1.3558; % Maximum Torque
related to max continuous current (N*m)
mc_max_trq2=Pmax./mc_map_spd; % Maximum Torque related to maximum continuous power (N*m)
[mc_corner_spd,mc_corner_pt,index]=polyxpoly(mc_map_spd,mc_max_trq1,mc_map_spd,mc_max_trq2);
% Corner Parameters
mc_max_trq(1:index)=mc_max_trq1(1:index); % Torque values for low-speed maximum torque region
mc_max_trq(index+1:length(mc_map_spd))=mc_max_trq2(index+1:length(mc_map_spd)); % Torque
values for maximum power region
mc_max_gen_trq=-1*mc_max_trq; % Maximum Regenerating Torque (N*m)
%%%%%%%%%% END OF LIMITS %%%%%%%%%%

%%%%%%%%%% OTHER DATA %%%%%%%%%%
mc_inertia=1.3E-6*mc_trq_scale*mc_spd_scale; % Rotor's Rotational Inertia (kg*m^2)
mc_mass=(0.161+0.02)/2.2; % Mass of Motor and Controller (kg)
mc_overtrq_factor=1.5; % Maximum Over-Torque for intermittent operation only
mc_tau=0.04; % Time Constant (s)
%%%%%%%%%% END OF OTHER DATA %%%%%%%%%%
```

A.2.5 UAV Data File

```
% 30 lb UAV Data File:  UAV_30lb.m

%%%%%%%%%% FILE ID INFO %%%%%%%%%%
uav_description='30 lb UAV';
disp(['Data loaded:  UAV Description - ',uav_description]);
%%%%%%%%%% END OF FILE ID INFO %%%%%%%%%%

%%%%%%%%%% UAV DATA %%%%%%%%%%
% Battery pack data
ess_module_num=1; % Number of Battery Modules
ess_init_soc=0.98; % Initial State of Charge (%)
ess_max_ah_capacity=interp1(ess_tmp,ess_max_ah_cap,(air_temp+10)-273); % (Ah)

% Propeller data
% Reference:  NACA Tech Note #698
prop_mass=0.17; % Prop Mass (kg), Reference:  DynaThrust Props, 20x8

% Coefficient data for 18x10 prop (2 blades)
prop_diam=0.457; % Prop Diameter (m)
prop_inertia=1/12*prop_mass*(prop_diam)^2; % Inertia of Prop, assumes slender rod (kg*m^2)
prop_J=[[0:0.05:0.7] 1]; % Advance Ratio, J
prop_CT= [0.0888 0.0880 0.0861 0.0831 0.0791 0.0743 0.0687 0.0623 0.0551 0.0471 0.0384 0.0289
0.0188 0.0080 -0.0036 -0.0869]; % Propeller Thrust Coefficient
prop_eff=[0.0340 0.1186 0.2324 0.3389 0.4362 0.5233 0.5989 0.6617 0.7099 0.7403 0.7467 0.7172 0.6239
0.3881  0.0001  0.0001]; % Efficiency vs. J
```

```
prop_CP=prop_CT.*prop_J./prop_eff; % Propeller Power Coefficient
prop_CP(1)=prop_CP(2);
prop_CP(15)=0.005;
prop_CP(16)=-0.07;

% Mass data
uav_total_mass=13.6;  % Total Mass of original or hybrid-electric UAV (kg)
WF_empty=0.63; % Weight Fraction for empty weight
uav_empty_mass_org=WF_empty*uav_total_mass; % Empty Mass of original UAV (kg)
eng_mass_org=2; % Engine Mass of original UAV (kg), 4.5 lbs-U.S. Engine
fuel_mass_org=2.1; % Fuel mass of original UAV (kg), (74 oz)
ess_mass_org=0.5; % Mass of original Battery Pack or Generator (kg)
uav_payload_mass_org=uav_total_mass-fuel_mass_org-uav_empty_mass_org; % Payload Mass of original
UAV (kg)
uav_glider_mass=uav_empty_mass_org-eng_mass_org-ess_mass_org-prop_mass; % Glider Mass of
original UAV (kg)
clutch_mass=0.25; % Clutch Mass for HEUAV (kg)
fc_mass=1.13; % Engine Mass for HEUAV (kg), (40 oz)
fuel_mass=1.53; % Fuel Mass of HEUAV (kg), (54 oz)
uav_payload_mass=uav_total_mass-uav_glider_mass-fuel_mass-fc_mass-clutch_mass-
ess_module_mass*ess_module_num-mc_mass-prop_mass; % Payload Mass of HEUAV (kg)
uav_empty_mass=uav_total_mass-fuel_mass-uav_payload_mass; % Empty Mass of HEUAV (kg)

% Lift-Drag Polar
uav_S=1.48; % Wing Area (m^2)
uav_chord=0.319; % Wing Chord (m)
uav_AR=14.6; % Aspect Ratio (chord of 12 in)
uav_oseff=0.85; % Oswald Efficiency Factor
uav_Cdo=0.036; % Zero-Lift Drag Coefficient, parasite drag
uav_Cl=[0:0.01:10]; % Lift Coefficient vector
uav_Cd=uav_Cdo+(uav_Cl.^2)/(pi*uav_oseff*uav_AR); % Drag Coefficient
uav_Clmax=1.25; % NACA Airfoil 23012, E214, S2091, or SD7032

% Electrical Accessories data
acc_elec_pwr=75; % Power Draw of Accessories (W), Original:  75
acc_elec_eff=1.0; % Efficiency of Accessories

% Power Train Gear Reduction data
gr1_ratio=1.0; % Ratio of Gear Reduction for engine
gr2_ratio=3.7; % Ratio of Gear Reduction for electric motor
gr_loss=0;  % Power Loss in Gear Reduction (W)
%%%%%%%%%% END OF UAV DATA %%%%%%%%%%
```

A.3 Controller Algorithm MATLAB Code

Off-line controller algorithms were written in MATLAB code. The two-input algorithm is intended to duplicate the rule-based controller. The three-input input algorithm has an additional input (i.e. battery SOC) and incorporates an objective function that is minimized.

A.3.1 Two-Input Algorithm

```
% Generate commanded engine torque using engine and motor efficiency maps
close all; % Close figures
clear all; % Clear variables
pack; % Clean up memory

% Load engine data and plot efficiency
HEUAV=1;
disp(' ');
disp('Select a specific engine:');
disp('  1:  FC_Q100 or PC125->2-Stroke Gasoline Engine, Scaled');
disp('  2:  GX31->4-Stroke Gasoline Engine');
fc_sel=input('Enter your selection:  ');
switch fc_sel
  case 1
    FC_Q100; % Quadra Aerrow 100 cm^3 2-Stroke Engine, from Ferguson
    %FC_PC125; % Piaggio Cosa 125 cm^3 2-Stroke Engine, from Heywood and Sher
  case 2
    FC_HondaGX31; % Honda GX31 4-Stroke Gasoline Engine
  case 3
    disp('Not a valid selection.');
end
figure; colormap([0 0 0]);
[Teng,weng]=meshgrid(fc_map_trq, fc_map_spd);
C=mesh(weng*(30/pi),Teng,fc_eff_map*100);
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Torque (N\cdotm)'); zlabel('Efficiency (%)');
title('Engine Efficiency Map');
%print -depsc2 -tiff -r600 ICE_Eff

% Load motor data and plot efficiency
MC_Aveox2739_3Y;
gr=3.7; % Gear Ratio
figure; colormap([0 0 0]);
[Tmot,wmot]=meshgrid(mc_map_trq, mc_map_spd);
C=mesh(wmot*(30/pi),Tmot,mc_eff_map*100);
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Torque (N\cdotm)'); zlabel('Efficiency (%)');
title('Motor Efficiency Map');
%print -depsc2 -tiff -r600 Motor_Eff

% Generate commanded torque curves for engine/motor combination
input1=transpose([190:10:880]); % Speed (propeller) map for input space (rad/sec)
input2=transpose([0:0.05:2.5]); % Torque (commanded) map for input space (N*m)
quant=[length(input1) length(input2)]; % Number of cells in each direction
% Compute hyperplane
eng_Torque=zeros(quant(1),quant(2)); % Pre-allocate memory
mot_Torque=zeros(quant(1),quant(2)); % Pre-allocate memory
for (i1=1:quant(1))
  for (i2=1:quant(2))
    eng_IOL_T=interp1(fc_map_spd,fc_IOL_trq,input1(i1));
    if (input2(i2)<eng_IOL_T)
      eng_T=input2(i2); % Engine torque less than IOL torque
      if(input2(i2)<0)
```

```
            eng_T=0;
          end
        else
          eng_T=eng_IOL_T; % Engine torque equal to IOL torque
        end
        mot_T=input2(i2)-eng_T; % Torque required from motor at prop
        if ((mot_T/gr)>interp1(mc_map_spd,mc_max_trq,gr*input1(i1)))
          mot_T=gr*interp1(mc_map_spd,mc_max_trq,gr*input1(i1)); % Set motor torque to max torque
          eng_T=input2(i2)-mot_T;
        end
        eng_Torque(i1,i2)=eng_T; % Engine torque at prop
        mot_Torque(i1,i2)=mot_T; % Motor torque at prop
    end
  end

  % Plot commanded engine torque
  [T,w]=meshgrid(transpose(input2), transpose(input1));
  figure; colormap([0 0 0]);
  C=mesh(w*(30/pi),T,eng_Torque);
  xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N\cdotm)'); zlabel('Commanded Engine Torque
  (N\cdotm)');
  set(gca,'GridLineStyle','--');
  title('Commanded Engine Torque');
  %print -depsc2 -tiff -r600 Trq_Cmd_ICE

  % Plot commanded motor torque
  figure; colormap([0 0 0]);
  C=mesh(w*(30/pi),T,mot_Torque);
  xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N\cdotm)'); zlabel('Commanded Motor Torque at
  Propeller (N\cdotm)');
  set(gca,'GridLineStyle','--');
  title('Commanded Motor Torque');
  %print -depsc2 -tiff -r600 Trq_Cmd_EM

  % Save commanded engine torque data to a file
  count=0;
  Trq=zeros(quant(1)*quant(2),3); % Pre-allocate memory
  for (i1=1:quant(1))
    for (i2=1:quant(2))
      count=count+1;
      Trq(count,:)=[input1(i1) input2(i2) eng_Torque(i1,i2)];
    end
  end

  % Write data to files
  if (fc_sel==1)
    save Trq_Cmd_ICE_EM_2S_data.txt Trq -ascii -tabs;
    copyfile('Trq_Cmd_ICE_EM_2S_data.txt','Trq_Cmd_ICE_EM_2S_train.txt');
    copyfile('Trq_Cmd_ICE_EM_2S_data.txt','Trq_Cmd_ICE_EM_2S_test.txt');
  elseif (fc_sel==2)
    save Trq_Cmd_ICE_EM_4S_data.txt Trq -ascii -tabs;
    copyfile('Trq_Cmd_ICE_EM_4S_data.txt','Trq_Cmd_ICE_EM_4S_train.txt');
    copyfile('Trq_Cmd_ICE_EM_4S_data.txt','Trq_Cmd_ICE_EM_4S_test.txt');
  else
    disp('Data not saved.')
  end
```

A.3.2 Three-Input Algorithm

```
% Generate commanded engine torque using engine, motor, and battery efficiency maps
close all; % Close figures
clear all; % Clear variables
pack; % Clean up memory

% Load engine data
HEUAV=1;
disp(' ');
disp('Select a specific engine:');
disp('  1:  FC_Q100 or PC125->2-Stroke Gasoline Engine, Scaled');
disp('  2:  GX31->4-Stroke Gasoline Engine');
fc_sel=input('Enter your selection:  ');
switch fc_sel
   case 1
      FC_Q100; % Quadra Aerrow 100 cm^3 2-Stroke Engine, from Ferguson
      %FC_PC125; % Piaggio Cosa 125 cm^3 2-Stroke Engine, from Heywood and Sher
   case 2
      FC_HondaGX31; % Honda GX31 4-Stroke Gasoline Engine
   case 3
      disp('Not a valid selection.');
end

% Select charging strategy
disp(' ');
disp('Select a charging strategy:');
disp('  1:  Charge-Sustaining');
disp('  2:  Charge-Depletion');
chg_strategy=input('Enter your selection:  ');

% Add zero torque row for interpolation
fc_map_trq=[0 fc_map_trq];
[m,n]=size(fc_eff_map);
fc_eff_map=[zeros(m,1) fc_eff_map];

% Plot engine efficiency
figure; colormap([0 0 0]);
[weng,Teng]=ndgrid(fc_map_spd,fc_map_trq);
C=mesh(weng*(30/pi),Teng,fc_eff_map*100);
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Torque (N\cdotm)'); zlabel('Efficiency (%)');
title('Engine Efficiency Map');
%print -depsc2 -tiff -r600 ICE_Eff

% Load motor data and plot efficiency
MC_Aveox2739_3Y;
gr=3.7; % Gear Ratio
figure; colormap([0 0 0]);
[wmot,Tmot]=ndgrid(mc_map_spd,mc_map_trq);
C=mesh(wmot*(30/pi),Tmot,mc_eff_map*100);
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Torque (N\cdotm)'); zlabel('Efficiency (%)');
title('Motor Efficiency Map');
%print -depsc2 -tiff -r600 Motor_Eff
```

```
% Load battery file and determine battery efficiency
ESS_UBI2590_Parallel;
ambient_temp=(288+10)-273; % Ambient Temperature at sea level (C)
ess_module_num=1; % Number of Battery Modules
ess_max_ah_capacity=interp1(ess_tmp,ess_max_ah_cap,ambient_temp); % (Ah)

% Determine battery efficiency
SOC=[0:0.1:1];
current=[-35:5:50];
ess_eff_map=zeros(length(SOC),length(current)); % Pre-allocate memory
for (i=1:length(SOC)) % SOC loop
  ess_init_soc=SOC(i);
  for (j=1:length(current)) % Current loop
    [T,X,Y]=sim('Battery_Model',[],'',[0.1,current(j)]);
    Voltage(i,j)=Y(length(Y),1);
    Voc(i,j)=Y(length(Y),3);
    if (current(j)<0)
      ess_eff_map(i,j)=Voc(i,j)/Voltage(i,j);
    else
      ess_eff_map(i,j)=Voltage(i,j)/Voc(i,j);
    end
  end
end

% Plot battery efficiency
figure; colormap([0 0 0]);
C=mesh(current,SOC*100,ess_eff_map*100);
set(gca,'GridLineStyle','--');
xlabel('Current (A)');ylabel('SOC (%)');zlabel('Efficiency (%)');
title('Battery Efficiency Map');
%print -depsc2 -tiff -r600 Battery_Eff

% Generate commanded torque curve from engine/motor/battery efficiency maps
input1=transpose([190:20:890]); % Speed (propeller) map for input space (rad/sec)
input2=transpose([-0.1:0.05:2.5]); % Torque (commanded) map for input space (N*m)
input3=transpose(SOC); % SOC map for input space
quant=[length(input1) length(input2) length(input3)]; % Number of cells in each direction
eng_Torque=zeros(quant(1),quant(2),quant(3)); % Pre-allocate memory
mot_Torque=zeros(quant(1),quant(2),quant(3)); % Pre-allocate memory

% Parameters for the run
im='linear'; % Interpolation Method

% Compute hyper-plane data used to train CMAC neural network
mot_dischg_eff=interp2(Tmot,wmot,mc_eff_map,0.1,1400,im); % EV operation at endurance speed
for (i1=1:quant(1))
  disp(i1);
  eng_IOL_T=interp1(fc_map_spd,fc_IOL_trq,input1(i1),im);
  eng_max_T=interp1(fc_map_spd,fc_max_trq,input1(i1),im);
  mot_max_T=interp1(mc_map_spd,mc_max_trq,gr*input1(i1),im);
  for (i2=1:quant(2))
    for (i3=1:quant(3))
      bat_dischg_eff=interp2(current,SOC,ess_eff_map,17.0,input3(i3),im); % Includes 5 A for acc.
      if (fc_sel==1) % 2-Stroke
        if (chg_strategy==1) % Charge-Sustaining
```

```
          alpha=10+10*(1-input3(i3)); % variable to penalize electricity use
          beta=0.05*input3(i3); % variable to penalize recharging
        else % Charge-Depletion
          alpha=6.5+1*(1-input3(i3)); % variable to penalize electricity use
          beta=5*input3(i3); % variable to penalize recharging
        end
      else % 4-Stroke
        if (chg_strategy==1) % Charge-Sustaining
          alpha=4+4*(1-input3(i3)); % variable to penalize electricity use
          beta=0.3*input3(i3); % variable to penalize recharging
        else % Charge-Depletion
          alpha=3+1*(1-input3(i3)); % variable to penalize electricity use
          beta=2*input3(i3); % variable to penalize recharging
        end
      end
      % Demanded torque less than zero
      if (input2(i2)<0)
        % Path 3
        eng_T_step=[0.01:0.02:eng_IOL_T];
        path1_eff=interp2(Teng,weng,fc_eff_map,eng_T_step,input1(i1),im);
        T_avail=input2(i2)-eng_T_step; % Torque Available to charge at prop
        mot_chg_eff=interp2(Tmot,wmot,mc_eff_map,T_avail/gr,gr*input1(i1),im);
        charge_current=input1(i1).*T_avail.*mot_chg_eff./16.7; % Could substitute in Voltage
        GN=isfinite(charge_current);
        bat_chg_eff=interp2(current,SOC,ess_eff_map,charge_current(GN),input3(i3),im);
        path3_eff=mot_chg_eff(GN).*bat_chg_eff.*bat_dischg_eff.*mot_dischg_eff; % Path 3 Eff.
        Pmot_chg=input1(i1)*eng_T_step(GN)./path1_eff(GN)-input1(i1)*(-T_avail(GN)).*path3_eff;
% Power for Path 3
        % Calculate objective function
        J=beta*Pmot_chg;
        [x,index]=min(J);
        eng_T_step=eng_T_step(GN);
        eng_T=eng_T_step(index);
clear('eng_T_step','path1_eff','T_avail','mot_chg_eff','charge_current','GN','bat_chg_eff','path3_eff','Peng','
Pmot_chg','J','x','index','eng_T_step');
      % Demanded torque equal to zero
      elseif (input2(i2)==0)
        eng_T=eng_Torque(i1,i2-1,i3);
      % Demanded torque greater than zero but less than max engine torque
      elseif (input2(i2)<=eng_max_T)
        % Engine torque less than the demanded torque
        eng_T_step1=[0.01:0.02:input2(i2)];
        % Path 1
        path1_eff=interp2(Teng,weng,fc_eff_map,eng_T_step1,input1(i1),im); % Path 1 Eff.-ICE only
        % Path 2
        mot_T_step=input2(i2)-eng_T_step1;
        mot_eff=interp2(Tmot,wmot,mc_eff_map,mot_T_step/gr,gr*input1(i1),im);
        GN1=isfinite(mot_eff);
        dischg_current=input1(i1).*mot_T_step(GN1)./(mot_eff(GN1).*14.4); % 14.4 V is nominal
voltage
        bat_eff=interp2(current,SOC,ess_eff_map,dischg_current+5,input3(i3),im); % Includes 5 A for
accessories
        path2_eff=bat_eff.*mot_eff(GN1);
        Peng1=input1(i1)*eng_T_step1(GN1)./path1_eff(GN1); % Power for Path 1
        Pmot1=input1(i1)*mot_T_step(GN1)./path2_eff; % Power for Path 2
        Pmot_chg1=zeros(1,sum(GN1));
        clear('path1_eff','mot_T_step','mot_eff','dischg_current','bat_eff','path2_eff');
        % Engine torque more than the demanded torque
        eng_T_step2=[input2(i2):0.02:eng_max_T];
        % Path 1
        path1_eff=interp2(Teng,weng,fc_eff_map,eng_T_step2,input1(i1),im); % Path 1 Eff.-ICE only
        % Path 3
        eng_T_avail=input2(i2)-eng_T_step2; % Torque Available to charge at prop
        mot_chg_eff=interp2(Tmot,wmot,mc_eff_map,eng_T_avail/gr,gr*input1(i1),im);
        charge_current=input1(i1).*eng_T_avail.*mot_chg_eff./16.7; % Could substitute in Voltage
        GN2=isfinite(charge_current);
        bat_chg_eff=interp2(current,SOC,ess_eff_map,charge_current(GN2),input3(i3),im);
        path3_eff=mot_chg_eff(GN2).*bat_chg_eff.*bat_dischg_eff.*mot_dischg_eff; % Path 3 Eff.
        Peng2=input1(i1)*input2(i2)./path1_eff(GN2); % Power for Path 1
        Pmot_chg2=input1(i1)*(-eng_T_avail(GN2))./path1_eff(GN2)-input1(i1)*
(-eng_T_avail(GN2)).*path3_eff; % Power for Path 3
        Pmot2=zeros(1,sum(GN2));
        % Calculate objective function
        eng_T_step=[eng_T_step1(GN1) eng_T_step2(GN2)];
        Peng=[Peng1 Peng2];
        Pmot=[Pmot1 Pmot2];
        Pmot_chg=[Pmot_chg1 Pmot_chg2];
        J=Peng+alpha*Pmot+beta*Pmot_chg;
        [x,index]=min(J);
        eng_T=eng_T_step(index);
clear('eng_T_step1','eng_T_step2','path1_eff','eng_T_avail','mot_chg_eff','charge_current','GN1','GN2','bat
_chg_eff','path3_eff','J','x','index','eng_T_step','Peng1','Peng2','Pmot1','Pmot2','Pmot_chg1','Pmot_chg2','Pe
ng','Pmot','Pmot_chg');
      % Demanded torque greater than max engine torque
      else
        eng_T_step=[0.01:0.02:eng_max_T];
        % Path 1
        path1_eff=interp2(Teng,weng,fc_eff_map,eng_T_step,input1(i1),im); % Path 1 Eff.-ICE only
        % Path 2
        mot_T_step=input2(i2)-eng_T_step;
        mot_eff=interp2(Tmot,wmot,mc_eff_map,mot_T_step/gr,gr*input1(i1),im);
        GN=isfinite(mot_eff);
        dischg_current=input1(i1).*mot_T_step(GN)./(mot_eff(GN).*14.4); % 14.4 V is the nominal
voltage
        bat_eff=interp2(current,SOC,ess_eff_map,dischg_current+5,input3(i3),im); % Includes 5 A for
accessories
        path2_eff=bat_eff.*mot_eff(GN);
        Peng=input1(i1)*eng_T_step(GN)./path1_eff(GN); % Power for Path 1
        Pmot=input1(i1)*mot_T_step(GN)./path2_eff; % Power for Path 2
        J=Peng+alpha*Pmot;
        [x,index]=min(J);
        eng_T_step=eng_T_step(GN);
        eng_T=eng_T_step(index);
clear('path1_eff','mot_T_step','mot_eff','dischg_current','bat_eff','path2_eff','Peng','Pmot','J','x','index','GN','
eng_T_step');
      end

      mot_T=input2(i2)-eng_T; % Torque required from motor at prop

      % Check motor max torque
      if ((mot_T/gr)>mot_max_T)
        mot_T=gr*mot_max_T; % Set motor torque to max torque
```

```
            eng_T=input2(i2)-mot_T;
            if eng_T>eng_max_T
               eng_T=eng_max_T;
            end
         end

         % Keep motor out of inefficient region
         if (abs(mot_T)<(gr*0.02))
            dT=mot_T;
            mot_T=gr*0.02;
            eng_T=eng_T-(gr*0.02-dT);
         end

         eng_Torque(i1,i2,i3)=eng_T; % Engine torque at prop
         mot_Torque(i1,i2,i3)=mot_T; % Motor torque at prop
      end
   end
end

% Perform interpolation to smooth data
[wcg,Tcg,SOCcg]=ndgrid(transpose(input1),transpose(input2),transpose(input3));
w=transpose([190:10:880]);
T=transpose([0:0.05:2.5]);
SOC=[0.02:0.02:1];
[wfg,Tfg,SOCfg]=ndgrid(w,T,SOC);
Trq1=zeros(length(w),length(T),length(SOC)); % Pre-allocate memory
Trq1=interpn(wcg,Tcg,SOCcg,eng_Torque,wfg,Tfg,SOCfg,'*linear');

% Plot commanded engine torque (SOC=10%)
figure; colormap([0 0 0]);
[w2D,T2D]=ndgrid(w,T);
mesh(w2D*(30/pi),T2D,Trq1(:,:,5));
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=10%')

% Plot commanded engine torque (SOC=25%)
figure; colormap([0 0 0]);
mesh(w2D*(30/pi),T2D,Trq1(:,:,12));
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=25%');

% Plot commanded engine torque (SOC=50%)
figure; colormap([0 0 0]);
mesh(w2D*(30/pi),T2D,Trq1(:,:,25));
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=50%');

% Plot commanded engine torque (SOC=75%)
figure; colormap([0 0 0]);
mesh(w2D*(30/pi),T2D,Trq1(:,:,37));
```

```
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=75%');

% Plot commanded engine torque (SOC=90%)
figure; colormap([0 0 0]);
mesh(w2D*(30/pi),T2D,Trq1(:,:,45));
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=90%');

% Plot commanded engine torque (SOC=100%)
figure; colormap([0 0 0]);
mesh(w2D*(30/pi),T2D,Trq1(:,:,50));
set(gca,'GridLineStyle','--');
xlabel('\omega (rpm)'); ylabel('Total Desired Torque (N \cdotm)'); zlabel('Commanded Engine Torque
(N\cdotm)');
title('Commanded Engine Torque, SOC=100%');

% Save commanded engine torque data to a file
count=0;
quant=[length(w) length(T) length(SOC)]; % Number of cells in each direction
Trq2=zeros(quant(1)*quant(2)*quant(3),4); % Pre-allocate memory
for (i1=1:quant(1))
  for (i2=1:quant(2))
    for (i3=1:quant(3))
      count=count+1;
      Trq2(count,:)=[w(i1) T(i2) SOC(i3) Trq1(i1,i2,i3)];
    end
  end
end

% Write data to files
if (fc_sel==1) % 2-Stroke Engine
  if (chg_strategy==1) % Charge-Sustaining
    save Trq_Cmd_ICE_EM_SOC_2S_CS_data.txt Trq2 -ascii -tabs
    copyfile('Trq_Cmd_ICE_EM_SOC_2S_CS_data.txt','Trq_Cmd_ICE_EM_SOC_2S_CS_train.txt');
    copyfile('Trq_Cmd_ICE_EM_SOC_2S_CS_data.txt','Trq_Cmd_ICE_EM_SOC_2S_CS_test.txt');
  else % Charge-Depletion
    save Trq_Cmd_ICE_EM_SOC_2S_CD_data.txt Trq2 -ascii -tabs
    copyfile('Trq_Cmd_ICE_EM_SOC_2S_CD_data.txt','Trq_Cmd_ICE_EM_SOC_2S_CD_train.txt');
    copyfile('Trq_Cmd_ICE_EM_SOC_2S_CD_data.txt','Trq_Cmd_ICE_EM_SOC_2S_CD_test.txt');
  end
elseif (fc_sel==2) % 4-Stroke Engine
  if (chg_strategy==1) % Charge-Sustaining
    save Trq_Cmd_ICE_EM_SOC_4S_CS_data.txt Trq2 -ascii -tabs
    copyfile('Trq_Cmd_ICE_EM_SOC_4S_CS_data.txt','Trq_Cmd_ICE_EM_SOC_4S_CS_train.txt');
    copyfile('Trq_Cmd_ICE_EM_SOC_4S_CS_data.txt','Trq_Cmd_ICE_EM_SOC_4S_CS_test.txt');
  else % Charge-Depletion
    save Trq_Cmd_ICE_EM_SOC_4S_CD_data.txt Trq2 -ascii -tabs
    copyfile('Trq_Cmd_ICE_EM_SOC_4S_CD_data.txt','Trq_Cmd_ICE_EM_SOC_4S_CD_train.txt');
    copyfile('Trq_Cmd_ICE_EM_SOC_4S_CD_data.txt','Trq_Cmd_ICE_EM_SOC_4S_CD_test.txt');
  end
else
```

```
  disp('Data not saved.')
end
```

A.4 CMAC ANN C++ Code

The original CMAC code was developed at Clemson by Professor James K. Peterson [159]. For the dissertation research, the code was formatted to allow it to run in a Microsoft Visual C++ environment. Other modifications include the addition of improved displacement vectors and improved cell width calculations for the CMAC structure. The entire CMAC code with the modifications is listed in this section. An input file is used to select options such as the function to approximate, type of engine, and type of charging strategy.

A.4.1 Input File

```
// CMAC_Function.cpp: Defines the entry point for the console application.

#include <stdafx.h>
#include "cmac.h"

#define DES_FUNCT (3) // 0-simple function, 1-sine function, 2-2 input controller, 3-3 input controller
#define ENGINE_TYPE (2) // 2-2-Stroke, 4-4-Stroke
#define CHG_STRATEGY (2) // 1-Charge-Sustaining, 2-Charge-Depletion
#define STOP_TOL (1.0e-4)
#define RUN_MAX (100)
#define DISPLAY_FREQ (10)
#define A_MATRIX (0) // 0-do not save A matrix, 1-save A matrix

int main(int argc,char *argv[])
{
    // declare and initialize variables
    int echo_input,do_train;
    int i,k,count,in_size,out_size,run,train_size,test_size,levels,A_value;
    float **in_train,**in_test,**out_train,**out_test,rms_train,rms_test;
    float u,v,w,x,y,z,rms_test2,*values;
    float **in_test2,**out_test2;
    char *file_name,*source_file,*destination_file,*A_matrix_file;
    FILE *in_file_pointer,*out_file_pointer,*fd,*A_file_pointer;

    echo_input=1;
    do_train=1;

    // set up CMAC architecture
    if(ENGINE_TYPE==2)
    {
        if (DES_FUNCT==1) file_name="Sine_Function.cfg";
        else if(DES_FUNCT==2) file_name="Trq_Cmd_ICE_EM_2S.cfg";
        else if(DES_FUNCT==3)
        {
            if(CHG_STRATEGY==1) file_name="Trq_Cmd_ICE_EM_SOC_2S_CS.cfg";
            else file_name="Trq_Cmd_ICE_EM_SOC_2S_CD.cfg";
        }
        else file_name="Simple_Function.cfg";
    }
    else
    {
        if (DES_FUNCT==1) file_name="Sine_Function.cfg";
        else if(DES_FUNCT==2) file_name="Trq_Cmd_ICE_EM_4S.cfg";
        else if(DES_FUNCT==3)
        {
            if(CHG_STRATEGY==1) file_name="Trq_Cmd_ICE_EM_SOC_4S_CS.cfg";
            else file_name="Trq_Cmd_ICE_EM_SOC_4S_CD.cfg";
        }
        else file_name="Simple_Function.cfg";
    }
    CMAC cmac(file_name);
    cmac.read_training_data(&in_train,&out_train);
    cmac.read_testing_data(&in_test,&out_test);
    cmac.initialize();
    cmac.number();

    // echo parameters to the screen
    if(echo_input) cmac.echo();

    // get training and test size
    cmac.get_train_size(&train_size).get_test_size(&test_size);

    // train the CMAC neural network, if desired
    if(do_train)
    {
        printf("\n\nTraining the CMAC neural network.  Please be patient:)\n");

        if((fd=fopen("RMS_data.txt","w"))==NULL)
        {
            printf("\nCan't open file %s\n","Training RMS");
            exit(0);
        } // save training rms data

        for(run=0;run<RUN_MAX;++run)
        {
            // compute rms of CMAC neural net compared to training set
            rms_train=cmac.compute_rms(train_size,in_train,out_train);
            // compute rms of CMAC neural net compared to test set
            rms_test=cmac.compute_rms(test_size,in_test,out_test);
            if(rms_train>STOP_TOL)
            {
                cmac.train(1,train_size,in_train,out_train);
                fprintf(fd,"%14.10f \n",rms_train);
                if(run%DISPLAY_FREQ==0)
                {
                    printf("rms train[%3d]=%12.6e  ",run,rms_train);
                    printf("rms test[%3d]=%12.6e\n",run,rms_test);
            }
        } // rms train>STOP_TOL loop
    } // run loop
    printf("rms train[%3d]=%12.6e ",run,rms_train);
    printf("rms test[%3d]=%12.6e\n",run,rms_test);
    cmac.write_wts();
    fclose(fd);
}
else
{
    run=0;
    // compute rms of CMAC neural net compared to training set
    rms_train=cmac.compute_rms(train_size,in_train,out_train);
    // compute rms of CMAC neural net compared to test set
    rms_test=cmac.compute_rms(test_size,in_test,out_test);
    printf("rms train[%3d]=%12.6e ",run,rms_train);
    printf("rms test[%3d]=%12.6e\n",run,rms_test);
}

// name files
printf("\n\n");
if(ENGINE_TYPE==2)
{
    if (DES_FUNCT==1) source_file="Sine_Function_data.txt";
    else if(DES_FUNCT==2) source_file="Trq_Cmd_ICE_EM_2S_data.txt";
    else if(DES_FUNCT==3)
    {
        if(CHG_STRATEGY==1) source_file="Trq_Cmd_ICE_EM_SOC_2S_CS_data.txt";
        else source_file="Trq_Cmd_ICE_EM_SOC_2S_CD_data.txt";
    }
    else source_file="Simple_Function_data.txt";
}
else
{
    if (DES_FUNCT==1) source_file="Sine_Function_data.txt";
    else if(DES_FUNCT==2) source_file="Trq_Cmd_ICE_EM_4S_data.txt";
    else if(DES_FUNCT==3)
    {
        if(CHG_STRATEGY==1) source_file="Trq_Cmd_ICE_EM_SOC_4S_CS_data.txt";
        else source_file="Trq_Cmd_ICE_EM_SOC_4S_CD_data.txt";
    }
    else source_file="Simple_Function_data.txt";
}
printf("source_file file=%s\n",source_file);

if(ENGINE_TYPE==2)
{
    if (DES_FUNCT==1) destination_file="Sine_Function_MATLAB_data.txt";
    else if(DES_FUNCT==2) destination_file="Trq_Cmd_ICE_EM_2S_MATLAB_data.txt";
    else if(DES_FUNCT==3)
    {
        if(CHG_STRATEGY==1)
            destination_file="Trq_Cmd_ICE_EM_SOC_2S_CS_MATLAB_data.txt";
        else destination_file="Trq_Cmd_ICE_EM_SOC_2S_CD_MATLAB_data.txt";
    }
    else destination_file="Simple_Function_MATLAB_data.txt";
```

```
}
else
{
    if (DES_FUNCT==1) destination_file="Sine_Function_MATLAB_data.txt";
    else if(DES_FUNCT==2) destination_file="Trq_Cmd_ICE_EM_4S_MATLAB_data.txt";
    else if(DES_FUNCT==3)
    {
        if(CHG_STRATEGY==1)
            destination_file="Trq_Cmd_ICE_EM_SOC_4S_CS_MATLAB_data.txt";
        else destination_file="Trq_Cmd_ICE_EM_SOC_4S_CD_MATLAB_data.txt";
    }
    else destination_file="Simple_Function_MATLAB_data.txt";
}
printf("destination_file=%s\n",destination_file);

// Open the source file
if((in_file_pointer=fopen(source_file,"r"))==NULL)
{
    printf("\nCan't open file %s\n",source_file);
    exit(1);
}

// Open the output file
if((out_file_pointer=fopen(destination_file,"w"))==NULL)
{
    printf("\nCan't open file $s\n",destination_file);
    exit(1);
}

// initialize counters
count=0;

// count the number of samples in the source file
if(DES_FUNCT==1) while(fscanf(in_file_pointer,"%f %f %f",&x,&y,&z)!=EOF) ++count;
else if(DES_FUNCT==2) while(fscanf(in_file_pointer,"%f %f %f",&x,&y,&z)!=EOF) ++count;
else if(DES_FUNCT==3) while(fscanf(in_file_pointer,"%f %f %f %f",&u, &v,&w,&x)!=EOF)
    ++count;
else while(fscanf(in_file_pointer,"%f %f %f",&x,&y,&z)!=EOF) ++count;
fclose(in_file_pointer);
printf("Number of samples in input file=%d\n\n\n",count);
printf("Number of samples in output file=%d\n",count);

// reopen the source file with read only status
if((in_file_pointer=fopen(source_file,"r"))==NULL)
{
    printf("\nCan't open file %s\n",source_file);
    exit(1);
}
fclose(in_file_pointer);

// set up CMAC architecture and read in weights
if(ENGINE_TYPE==2)
{
    if(DES_FUNCT==1) file_name="Sine_Function_plot.cfg";
    else if(DES_FUNCT==2) file_name="Trq_Cmd_ICE_EM_2S_plot.cfg";
    else if(DES_FUNCT==3)
```

```
        {
            if(CHG_STRATEGY==1) file_name="Trq_Cmd_ICE_EM_SOC_2S_CS_plot.cfg";
            else file_name="Trq_Cmd_ICE_EM_SOC_2S_CD_plot.cfg";
        }
        else file_name="Simple_Function_plot.cfg";
    }
    else
    {
        if(DES_FUNCT==1) file_name="Sine_Function_plot.cfg";
        else if(DES_FUNCT==2) file_name="Trq_Cmd_ICE_EM_4S_plot.cfg";
        else if(DES_FUNCT==3)
        {
            if(CHG_STRATEGY==1) file_name="Trq_Cmd_ICE_EM_SOC_4S_CS_plot.cfg";
            else file_name="Trq_Cmd_ICE_EM_SOC_4S_CD_plot.cfg";
        }
        else file_name="Simple_Function_plot.cfg";
    }
    CMAC cmac2(file_name);
    cmac2.read_training_data(&in_test2,&out_test2).initialize().number();
    cmac2.echo();

    /*
    Generate a file of test inputs and cmac outputs for graphing and for HEUAV model:
    Assume multiple inputs and 1 output.
    */

    printf("The instantiation of the CMAC object is finished.\n");
    cmac2.get_output_size(&out_size);
    cmac2.get_input_size(&in_size);
    values=new float[out_size];
    for(k=0;k<count;++k)
    {
        // save CMAC values to output file
        values=cmac2.cmac_eval(1,in_test2[k]);
        out_test2[k][0]=values[0];
        for(i=0;i<in_size;++i)
        {
            fprintf(out_file_pointer,"%8.4f ",in_test2[k][i]);
        }
        fprintf(out_file_pointer,"%8.4f\n",values[0]);
    }
    fclose(out_file_pointer);

    // save A matrix to a file, if desired
    if(A_MATRIX==1)
    {
        // name A matrix file
        if(ENGINE_TYPE==2)
        {
            if(DES_FUNCT==1) A_matrix_file="A_matrix_Sine_Function.txt";
            else if(DES_FUNCT==2) A_matrix_file="A_matrix_ICE_EM_2S.txt";
            else if(DES_FUNCT==3)
            {
                if(CHG_STRATEGY==1)A_matrix_file="A_matrix_ICE_EM_SOC_2S_CS.txt";
                else A_matrix_file="A_matrix_ICE_EM_SOC_2S_CD.txt";
            }
```

```
                else A_matrix_file="A_matrix_Simple_Function.txt";
            }
            else
            {
                if(DES_FUNCT==1) A_matrix_file="A_matrix_Sine_Function.txt";
                else if(DES_FUNCT==2) A_matrix_file="A_matrix_ICE_EM_4S.txt";
                else if(DES_FUNCT==3)
                {
                    if(CHG_STRATEGY==1)A_matrix_file="A_matrix_ICE_EM_SOC_4S_CS.txt";
                    else A_matrix_file="A_matrix_ICE_EM_SOC_4S_CD.txt";
                }
                else A_matrix_file="A_matrix_Simple_Function.txt";
            }
            printf("A_matrix_file=%s\n",A_matrix_file);

            // Open the A matrix file
            if((A_file_pointer=fopen(A_matrix_file,"w"))==NULL)
            {
                printf("\nCan't open file $s\n",A_matrix_file);
                exit(1);
            }

            // save the A matrix
            cmac2.get_levels(0,&levels);
            for(k=0;k<count;++k)
            {
                values=cmac2.cmac_eval(1,in_test2[k]);
                for(i=0;i<levels;++i)
                {
                    cmac2.get_A_value(0,i,&A_value);
                    fprintf(A_file_pointer,"%d ",A_value);
                }
            fprintf(A_file_pointer,"\n");
            }
            fclose(A_file_pointer);
        }

        // compute rms on model data
        rms_test2=cmac2.compute_rms(count,in_test2,out_test2);

        delete in_test2;
        delete out_test2;
        return(1);
}
```

A.4.2 CMAC ANN Program

/*
The header file for the CMAC class. A few auxiliary functions, primarily for allocating and deallocating two-dimensional arrays of various data types, are defined outside of the class in the file utility.h. Constants are defined in the file myconstants.h.

The CMAC class is designed to use a hashing function to calculate the working memory addresses from the virtual addresses. The code allows for a separate hash function to be used for each level of each output CMAC architecture. However, in practice, a single hash function is used

```
to initialize all the individual hash functions in the CMAC architecture.  Also, in practice
for the HEUAV application, no hashing is used.
*/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "my_constants.h"
#include "utility.h"

#define check (1)
#define diagnostic (0)
#define A_MATRIX (0)

typedef char file_name[MY_TEXT_SIZE];
typedef float *pfloat;
typedef int (* fhash)(int size,int *active_cells,int *num_intervals,int hash_size);

typedef struct
{
    char config_file[MY_TEXT_SIZE];
    char structure_file[MY_TEXT_SIZE];
    char training_file[MY_TEXT_SIZE];
    char testing_file[MY_TEXT_SIZE];
    char wts_init_file[MY_TEXT_SIZE];
    char wts_file[MY_TEXT_SIZE];
    int training_set_size;
    int testing_set_size;
    int in_dimensions;
    int out_dimensions;
    float learning_rate;
    float *min_x;
    float *max_x;
    float *input_resolution;
    file_name *file;
    int autosize;
    int echo_input;
    int do_training;
    int read_in_wts;
    float inflate_bottom;
    float inflate_top;
    float common_start;
    float common_end;
} CMACINPUT;

/*
The meaning of the given structure elements is explained below.  The value of MY_TEXT_SIZE
is set in cmac.h to be 256.

char config_file[MY_TEXT_SIZE]:
    Configuration file for the CMAC object instantiation, typically named name.cfg.
char structure_file[MY_TEXT_SIZE]:
    File typically called name_structure.txt and is the name of the file which contains the
    names of the files which initialize the structure of each output CMAC architecture.
char training_file[MY_TEXT_SIZE]:
```

File that stores the training data.
char testing_file[MY_TEXT_SIZE]:
File that stores the testing data.
char wts_init_file[MY_TEXT_SIZE]:
File that stores the weights from a previous training run.
char wts_file[MY_TEXT_SIZE]:
File in which the CMAC object parameters are stored, typically after they have been updated by the CMAC training process.
int training_set_size:
Number of data items in the training set.
int testing_set_size:
Number of data items in the testing set.
int in_dimensions:
Dimension of the input space.
int out_dimensions:
Dimension of the output space.
float learning_rate:
Learning rate used in the CMAC training algorithm.
float *min_x:
The CMAC architecture focuses on a segment of the ith input which will be denoted by [a_i,b_i]. The exact values of a_i and b_i are determined both by the training data and whether the data is autosized or uses a common start and end value. Once these values are set, a_i is stored in min_x[i].
float *max_x:
The CMAC architecture focuses on a segment of the ith input which will be denoted by [a_i,b_i]. The exact values of a_i and b_i are determined both by the training data and whether the data is autosized or uses a common start and end value. Once these values are set, b_i is stored in max_x[i].
float *input_resolution:
The resolution of the input space (currently calculated from the training data) for each input dimension.
file_name *file:
Vector that stores the names of the files that determine the architecture of the IOMAP structures for each output dimension. See the discussion for the class function structure_files().
int autosize:
This variable determines how the interval [a_i,b_i] is set up for the ith input dimension. Setting autosize to 1 allows for a stretching or shrinking of the actual values obtained via the training data. Setting autosize to 0 will force a common interval [a,b] to be used.
int echo_input:
A flag to allow for an automatic echoing of CMAC object parameters to the screen.
int do_training:
A flag to allow the CMAC architecture to train on the given training data.
int read_in_wts:
A flag to allow the CMAC object parameters to be read in from the external file setup.wts_init_file in order to initialize the CMAC object.
float inflate_bottom:
A fudge factor to expand or shrink the a_i value of the ith inputs interval.
float inflate_top:
A fudge factor to expand or shrink the b_i value of the ith inputs interval.
float common_start:
If a constant interval [a,b] is used for the input intervals, a is set to common_start.
float common_end:
If a constant interval [a,b] is used for the input intervals, b is set to common_end.

The basic computational unit of the CMAC architecture is the IOMAP structure. One IOMAP

```
structure is required for each output dimension in the CMAC architecture. Hence, the IOMAP
structure contains the information about the number of levels (layers), cell characteristics
such as offset and cell width, etc.
*/

typedef struct
{
    int levels;
    int hash_size_construct;
    float offset_size;
    float width;
    float **offset;
    float **cell_width;
    int *hash_size;
    fhash *hash_function;
    int **num_intervals;
    int *num_cells;
    int total_cells;
    int *active_cells;
    int *A_vector;
    int *working_address;
    float **working_memory;
    float output;
} IOMAP;

/*
The individual field elements have the following meaning:

int levels:
    Number of levels (layers, generalization factor), L, of cells used in the CMAC architecture.
int hash_size_construct:
    The hash size, H, used in the construction of the working memory.
float offset_size:
    The offset is based on the number of inputs, the input_resolution, and L so the offsets can
    be set as described in Appendix B in Brown and Harris. If desired, each level of cells can
    be offset from the previous level by the constant value O=W/L.
float width:
    The width is determined from the number of layers and the input resolution. If desired,
    the width, W, of the cells for this architecture can be set to a constant.
float **cell_width:
    A potentially different cell width for each level i and each input j may be required.
    Hence, a two-dimensional array of size Lxn is required. Thus, iomap.cell_width[i][j]
    is the cell width for the ith level and jth input.
float **offset:
    A potentially different offset for each level i and each input j may be required. Hence,
    a 2-D array of size Lxn is required where n is the dimension of the input space. Thus,
    iomap.offset[i][j] is the offset value for the ith level and jth input. If required, the
    offsets may be set to the improved offsets determined by Parks and Militzer and listed in
    Appendix B of Brown and Harris.
int *hash_size:
    A different hash size may be required for each level. Hence, iomap.hash_size is size L and
    iomap.hash_size[i] is the hash size for the ith level.
fhash *hash_function:
    A different hash function can be used for each level. Thus iomap.hash_function[i] is the
    hash function for the ith level.
int **num_intervals:
        The number of intervals (cells) for each level i and input j. Thus, iomap.num_intervals is
        size Lxn and iomap.num_intervals[i][j] gives the number of cells for level i and input j.
int *num_cells:
        The number of cells for each level i. Thus, iomap.num_cells is size L and iomap.num_cells[i]
        gives the number of cells for the ith level.
int total_cells:
        The number of total cells (basis functions) for a CMAC architecture.
int *active_cells:
        A vector that stores the active cell list for a level for a given data vector x as the
        array [a_0,...,a_(n-1)], where a_j is the index of the active cell for input j.
float A_vector:
        Association vector for data x-size is the number of levels. The entries are the basis
        functions that are active for data x.
int *working_address:
        Once the working address for data x for level i is computed via hashing, the address is
        stored in the vector iomap.working_address. Hence, iomap.working_address has size L and
        iomap.working_address[i] is the hashed address of data x for level i.
float **working_memory:
        Array that stores the values that are used to compute the output of the iomap architecture
        for a given data x. Hence, iomap.working_memory has L rows, where each row stands for a
        level. Row i corresponds then to level i and has size iomap.hash_size[i]. Thus,
        iomap.working_memory[i][j] gives the value of the CMAC weight for level i and memory
        position j.
float output:
        The scalar ouput of the CMAC IOMAP architecture.

The CMAC class functions appear below. Note that the CMAC is instantiated using private elements:
a CMACINPUT structure and a pointer to the IOMAP structure.
*/

class CMAC
{
     public:
     CMAC(char *config_file);
     CMAC(void);
     CMAC(const CMAC&);
     CMAC& CMAC::operator=(const CMAC&);
     ~CMAC(void);
     CMAC& config(char *config_file);
     CMAC& structure_files(void);
     CMAC& allocate_memory(void);
     CMAC& initialize(void);
     CMAC& virtual_memory(int k,float *x);
     CMAC& read_training_data(float ***in,float ***out);
     CMAC& read_testing_data(float ***in,float ***out);
     CMAC& write_wts(void);
     CMAC& read_wts(void);
     CMAC& echo(void);
     CMAC& get_input_size(int *input_size);
     CMAC& get_output_size(int *output_size);
     CMAC& get_train_size(int *train_size);
     CMAC& get_test_size(int *test_size);
     CMAC& get_levels(int which_output,int *levels);
     CMAC& get_number_intervals(int which_output,int which_level,int which_input,int *number);
     CMAC& get_total_cells(int which_output,int *total_cells);
     CMAC& get_A_value(int which_output,int which_cell,int *value);
```

```
        CMAC& get_min_x(int which_input,float *min);
        CMAC& get_max_x(int which_input,float *max);
        CMAC& get_width(int which_output,int which_level,int which_input,float *width);
        CMAC& get_working_address(int which_output,int *address);
        CMAC& load_input_size(int input_size);
        CMAC& load_output_size(int output_size);
        CMAC& load_train_size(int train_size);
        CMAC& load_test_size(int test_size);
        CMAC& number(void);
        float compute_rms(int sample_size,float **in,float **out);
        float *cmac_eval(int do_address,float *x);
        CMAC& train(int do_address,int sample_size,float **in,float **out);
        private:
        CMACINPUT setup;
        IOMAP *iomap;
};

/*
The item listed below is the prototype for the common hash function used in the instantiation
of the architecture.
*/

int hhash(int size,int *active_cells,int *num_intervals,int hash_size);

/*
Generic hash function for the CMAC architecture. The variable size stores the value of n
(number of inputs). The active_cells vector coming in represents the coarse encoding the CMAC
architecture gives to an input x. In other words, letting n denote the number of inputs and
a_i the active cell or subinterval that the ith input component resides in, the array
active_cells stores the active cell list [a_0,a_1,...,a_(n-1)] for a level. Let the number
of cells or subintervals for input component i be denoted by M_i. The full list of the number
of cells per input component has been placed in the array num_intervals=[M_0,M_1,...,M_(n-1)].
Then, the list [a_0,a_1,...,a_(n-1)] represents the location of an element in an n-dimensional
input space where the ith input component can take on at most M_i values. Alternately, an
n-dimensional array, B, can be visualized where the ith row has M_i elements. The list
[a_0,a_1,...,a_(n-1)] then represents the array element B_(a_0,a_1,...,a_(n-1)) in an array
with unequal row sizes. Now this ragged array B can be converted into an equivalent
contiguous block of memory space by linearizing the elements array address.
(see original documentation)
*/

int hhash(int size,int *active_cells,int *num_intervals,int hash_size)
{
    int i,work_address,accumulator;
    unsigned long virtual_address;

    virtual_address=0;
    accumulator=1;
    for(i=0;i<size;++i)
    {
        virtual_address=virtual_address*(unsigned long)accumulator
            +(unsigned long)active_cells[size-1-i];
        virtual_address=virtual_address%(unsigned long)hash_size;
        if(i<size-1) accumulator=num_intervals[size-i-2];
    }
    work_address=virtual_address%(unsigned long)hash_size;
    return(work_address);
}

/*
Explicit constructor for the CMAC class.  To use this function, type a source line like
CMAC S("application.cfg"); and all of the information stored in the file "application.cfg"
will be used in the instantiation of the CMAC object S.  The function simply calls the
configuration function CMAC& CMAC::config(char *config_file).
*/

CMAC::CMAC(char *config_file)
{
    printf("The explicit constructor was used to obtain the data below from the data file:\n");
    config(config_file);
}

/*
Default constructor for the CMAC class. To use this function, type a source line like CMAC T[4];
and at compile time the default constructor is called.  An example of the code use is as
follows:

CMAC S("application.cfg");
CMAC T[4];

for(i=0;i<4;++i)
{
    T[i]=S;
}

Line by line description of the code:
    1. The explicit constructor is used to initialize the CMAC object S using the information
    in the file "application.cfg". The file name "application.cfg" is stored as part of the
    CMAC object creation process in S.setup.config\file.

    2. The default constructor is called to instantiate four objects, T[4].

    3. The overloaded = is then used to build the CMAC object T[i] by first copying the
    configuration file S.setup.config_file into T[i].setup.config_file.  Then the configuration
    file, which is simply "application.cfg", is used in the class function config() to construct
    the new CMAC object T[i] that will share the values of S.  The overloaded = is automatically
    invoked by the syntax of this line.

There are some subtle points here.  During training of the CMAC architecture, critical parameters
are periodically written into a file whose name is stored in setup.wts_file.  Also, if the CMAC
architecture parameters are to be initialized using values stored in a file, the name of such an
initialization file is stored in setup.wts_init_file.  The names of these files are simply
copied when the class equal is invoked.  Hence, since it is not likely that different CMAC
architectures should share commom weight files and/or initialization files, a class function
is provided to set the names of these files.
*/

CMAC::CMAC(void)
{
    printf("The default constructor has been called.\n");
    iomap=NULL;
}
```

```
/*
Class copy constructor for the CMAC class.  If the following source code is used:

CMAC S("application.cfg");
CMAC T=S;

then the following sequence of events takes place:
    1. The explicit constructor is used to build the CMAC object S using the information in the
    file "application.cfg".  The file name "application.cfg" is stored as part of the CMAC
    object creation process in S.setup.config_file.

    2. The copy constructor is used to build the CMAC object T by first copying the config file
    S.setup.config_file into T.setup.config_file.  Then the configuration file, which is simply
    "application.cfg", is used in the class function config() to construct the new CMAC object T
    that will share the values of S.  The copy constructor is automatically invoked by the
    syntax of this line.
*/

CMAC::CMAC(const CMAC& S)
{
    strcpy(setup.config_file,S.setup.config_file);
    config(setup.config_file);
}

/*
Class = operator.  If the following source is used:

CMAC S("application.cfg");
CMAC T;
T=S;

then the following sequence of events takes place:
    1. The standard constructor is used to build the CMAC object S using the information in the
    file "application.cfg".  The file name "application.cfg" is stored as part of the CMAC
    object creation process in S.setup.config_file.

    2. The default constructor is used to instantiate CMAC object T with the iomap        private
    element set to NULL.

    3. The class = is then used to build the CMAC object T by first copying the config file
    S.setup.config_file into T.setup.config_file.  Then the configuration file, which is simply
    "application.cfg", is used in the class function config() to construct the new CMAC object T
    that will share the values of S.  The class = is automatically invoked by the syntax of this
    line.
*/

CMAC& CMAC::operator=(const CMAC& S)
{
    strcpy(setup.config_file,S.setup.config_file);
    config(setup.config_file);

    return *this;
}

/*
```

```
Default destructor for the CMAC object.  Note that the private elements of a CMAC object contain
only one pointer, IOMAP *iomap.  This pointer is explicitly freed in the destructor.
*/

CMAC::~CMAC(void)
{
    delete iomap;
}

/*
This function is the core of the constructors for the CMAC class.  When this function is called,
the appropriate allocations and initializations are performed.
*/

CMAC& CMAC::config(char *config_file)
{
    FILE *fd;
    char text[MY_TEXT_SIZE];

    // store the name of the configuration file in the IOMAP structure
    strcpy(setup.config_file,config_file);
    printf("%s\n",setup.config_file);

    if((fd=fopen(config_file,"r"))==NULL)
    {
        printf("\nCan't open file %s\n",config_file);
        exit(0);
    }

    // name of file containing structure information for all the IOMAP structures that will
    // comprise the full CMAC
    get_my_text(setup.structure_file,fd,1);

    // name of training file
    get_my_text(setup.training_file,fd,1);

    // name of testing file
    get_my_text(setup.testing_file,fd,1);

    // name of file that initializes the CMAC tunable parameters
    get_my_text(setup.wts_init_file,fd,1);

    // name of file to which the CMAC tunable paramters will be written to during the training
    // process
    get_my_text(setup.wts_file,fd,1);

    // training set size
    fscanf(fd,"%d",&(setup.training_set_size));
    printf("setup.training_set_size=%d",setup.training_set_size);
    get_my_text(text,fd,0);

    // testing set size
    fscanf(fd,"%d",&(setup.testing_set_size));
    printf("setup.testing_set_size=%d",setup.testing_set_size);
    get_my_text(text,fd,0);
```

```
// number of input dimensions
fscanf(fd,"%d",&(setup.in_dimensions));
printf("setup.in_dimensions=%d",setup.in_dimensions);
get_my_text(text,fd,0);

// number of output dimensions
fscanf(fd,"%d",&(setup.out_dimensions));
printf("setup.out_dimensions=%d",setup.out_dimensions);
get_my_text(text,fd,0);

// learning rate
fscanf(fd,"%f",&(setup.learning_rate));
printf("setup.learning_rate=%f",setup.learning_rate);
get_my_text(text,fd,0);

// echo input
fscanf(fd,"%d",&(setup.echo_input));
printf("setup.echo_input=%d",setup.echo_input);
get_my_text(text,fd,0);

// do_training
fscanf(fd,"%d",&(setup.do_training));
printf("setup.do_training=%d",setup.do_training);
get_my_text(text,fd,0);

// read_in_weights
fscanf(fd,"%d",&(setup.read_in_wts));
printf("setup.read_in_wts=%d",setup.read_in_wts);
get_my_text(text,fd,0);

// autosize
fscanf(fd,"%d",&(setup.autosize));
printf("setup.autosize=%d",setup.autosize);
get_my_text(text,fd,0);

if(setup.autosize==0) // do not autosize, use common_start and common_end
{
    fscanf(fd,"%f",&setup.common_start);
    printf("setup.common_start=%d",setup.common_start);
    get_my_text(text,fd,0);
    fscanf(fd,"%f",&setup.common_end);
    printf("setup.common_end=%d",setup.common_end);
    get_my_text(text,fd,0);
}
else // autosize using inflate_bottom and inflate_top
{
    // inflate_bottom
    fscanf(fd,"%f",&(setup.inflate_bottom));
    printf("setup.inflate_bottom=%f",setup.inflate_bottom);
    get_my_text(text,fd,0);
    // inflate_top
    fscanf(fd,"%f",&(setup.inflate_top));
    printf("setup.inflate_top=%f",setup.inflate_top);
    get_my_text(text,fd,0);
}
```

```
        // allocate space for vector containing minimum input dimensions
        setup.min_x=new float[setup.in_dimensions];

        // allocate space for vector containing maximum input dimensions
        setup.max_x=new float[setup.in_dimensions];

        // allocate space for vector containing input resolution
        setup.input_resolution=new float[setup.in_dimensions];

        // allocate memory for setup.out_dimensions files to contain each output component's CMAC
        // configuration information
        setup.file=new file_name[setup.out_dimensions];

        // allocate space for setup.out_dimensions IOMAP structures
        iomap=new IOMAP[setup.out_dimensions];

        structure_files();
        fclose(fd);

        return *this;
}

/*
For each output dimension in the CMAC architecture, there is a separate IOMAP structure. The
individual field elements of this structure are set using information stored in a file whose
name is stored in the private CMAC class element setup.file[n] of structure CMACINPUT. The main
configuration file for the CMAC architecture is usually called name.cfg where name is chosen to
be pertinent to the application. The first item inside this file is the name of the file which
itself contains the names of the files used to initialize the IOMAP structures. This file is
typically called name_structure.txt. The name name_structure.txt has already been stored in the
private CMAC class element setup.structure_file. Inside this file setup.structure_file is a
list of files with names of the form name_out0.cfg, name_out1.cfg, and so forth. The total
number of files is the same as the output dimension of the CMAC architecture. Hence, the file
containing the information to initialize the nth IOMAP structure is called name_outn.cfg.

The structure_files class function's purpose is to open the file setup.structure_file and
extract the individual IOMAP structure file names, one for each output dimension. These file
names are then stored in the field elements setup.file[].
*/

CMAC& CMAC::structure_files(void)
{
        int i;
        FILE *fd;

        // open structure_file
        if((fd=fopen(setup.structure_file,"r"))==NULL)
        {
            printf("\nCan't open file %s\n",setup.structure_file);
            exit(0);
        }

        // get CMAC structure file for IOMAP for ith output dimension
        for(i=0;i<setup.out_dimensions;++i)     get_my_text(setup.file[i],fd,1);

        fclose(fd);
```

```
    return *this;
}

/*
This function handles the primary allocation of memory for the CMAC object.  The individual
allocation tasks are carefully documented in the code below.
*/

CMAC& CMAC::allocate_memory(void)
{
    FILE *fd;
    char text[MY_TEXT_SIZE];
    int i,j;

    // The IOMAP for the ith output dimension is initialized using the file name stored in
    // setup.file[i].  The entries in setup.file[] are read in using the function
    // structure_file().
    for(j=0;j<setup.out_dimensions;++j)
    {
        if((fd=fopen(setup.file[j],"r"))==NULL)
        {
            printf("\nCan't open file %s\n",setup.file[j]);
            exit(0);
        }

        // number of levels
        fscanf(fd,"%d",&(iomap[j].levels));
        get_my_text(text,fd,1);

        // hash size
        fscanf(fd,"%d",&(iomap[j].hash_size_construct));
        get_my_text(text,fd,1);

        // offset-default size
        iomap[j].offset_size=(float)1.0/(float)iomap[j].levels;

        // width-default size
        fscanf(fd,"%f",&(iomap[j].width));
        get_my_text(text,fd,1);

        fclose(fd);

        // allocate space for iomap[j].offset, an array containing offsets for each level in
        // iomap[j].  This is a 2D matrix having iomap[j].levels rows and setup.in_dimensions
        // columns.
        get_2dmatrix_float(&(iomap[j].offset),iomap[j].levels,setup.in_dimensions);

        // allocate space for iomap[j].cell_width, an array containing cell
        // widths for each level in iomap[j].  This is a 2D matrix having iomap[j].levels rows
        // and setup.in_dimensions columns.
        get_2dmatrix_float(&(iomap[j].cell_width),iomap[j].levels,setup.in_dimensions);

        // allocate space for vector containing hash_size_construct for each level of iomap[j].
        iomap[j].hash_size=new int[iomap[j].levels];
        for(i=0;i<iomap[j].levels;++i) iomap[j].hash_size[i]=iomap[j].hash_size_construct;
        // allocate space for vector containing hash functions for each level of iomap[j].
        iomap[j].hash_function=new fhash[iomap[j].levels];

        // allocate space for iomap[j].working_memory, a 2D array storing working memory for
        // each level in iomap[j]. This is a 2D matrix having iomap[j].levels rows and each
        // row is length iomap[j].hash_size[i].
        iomap[j].working_memory=new pfloat[iomap[j].levels];
        for(i=0;i<iomap[j].levels;++i) iomap[j].working_memory[i]=new float[iomap[j].hash_size[i]];

        // allocate space for a a vector to temporarily store the computed address for iomap[j]
        iomap[j].working_address=new int[iomap[j].levels];

        // allocate space for iomap[j].num_intervals, a 2D array storing the number of intervals
        // for each level and each input component in iomap[j]. This is a 2D matrix having
        // iomap[j].levels rows and setup.in_dimensions columns.
        get_2dmatrix_int(&(iomap[j].num_intervals),iomap[j].levels,setup.in_dimensions);

        // allocate space for iomap[j].num_cells, a vector that stores the number of cells in
        // each layer.
        iomap[j].num_cells=new int[iomap[j].levels];

        // allocate space for iomap[j].active_cells, a vector that stores the active cell list
        // for a given input vector x.
        iomap[j].active_cells=new int[setup.in_dimensions];

        // allocate space for iomap[j].A_vector, a vector that stores the active cells for one
        // x value.
        iomap[j].A_vector=new int[iomap[j].levels];
    } // output dimension loop

    return *this;
}

/*
This function sets the initial values of many of the important CMAC object parameters. For each
output dimension, there is a corresponding IOMAP structure, iomap[i]. The architecture
specified by iomap[i] consists of L=iomap[i].levels.

Each IOMAP structure iomap[i] is essentially a CMAC architecture to model a scalar-valued
input/output mapping from an input space of dimension n=setup.in_dimensions. Each of these
architectures consists of L_i levels and on each of these levels the portion of the input space
on which the CMAC architecture focuses is divided up into hypercubes that function as binary
cells. Each of the hypercubes on level j is constructed from sides of length
w_jk=iomap[i].cell_width[j][k].

Each of the cell widths can be set in several ways. The width can be set using a
common value stored in the IOMAP structure, W=iomap[i].width. This is certainly not the
best solution for all applications. This approach subdivides the input space into hypercubes of
fixed side length. Another way is to set the widths so a cell in a layer covers as many
input lattice cells in one input dimension as there are levels, L. This is discussed in Brown
and Harris (pg 623). To enable this approach for setting cell widths, set OPTION to 1.

The hypercubes for each level j and input k are staggered using an offset strategy. All of the
offsets are set to a common value, if desired:
O_jk=iomap[i].offset[j][k]=O=iomap[i].offset_size.
The numerical value of the common offset parameter is set to be the reciprocal of the number of
```

```
levels L.  The offsets are then staggered for the kth input as follows:

O_0k=0.0
O_1k=w_1k*1/L=W/L
O_2k=2*w_2k*1/L=2W/L
O_3k=3*w_3k*1/L=3W/L ...
O_(L-1),k=(L-1)*w_(L-1),k*1/L=(L-1)W/L

Hence, an additional offset term would give an offset value of W and then the same offset
strategy is repeated.

If OPTION is set to 1, the offset is determined based on the input resolution and the tables in
Appendix B in Brown and Harris.
*/

#define OPTION (1)
CMAC& CMAC::initialize(void)
{
    int i,j,k;

    allocate_memory();
    printf("CMAC architecture data (example-cell widths) for each level:\n");
    for(i=0;i<setup.out_dimensions;++i)
    {
        for(j=0;j<iomap[i].levels;++j)
        {
            // initialize the hash function for each level of iomap[i]
            iomap[i].hash_function[j]=hhash;

            // initialize cell widths
            for(k=0;k<setup.in_dimensions;++k)
            {
                // cell width for level j, input k
                if(OPTION)
                {
                    iomap[i].cell_width[j][k]=float(iomap[i].levels)*setup.input_resolution[k];
                    printf("Cell width [output=%d,level=%d,input=%d]=%12.6f\n",i,j,k,
                        iomap[i].cell_width[j][k]);
                }
                else
                {
                    iomap[i].cell_width[j][k]=iomap[i].width;
                }
                // offset for level j and input k
                if(OPTION)
                {
            // Improved displacement vectors from Appendix B in Brown and Harris
            // are programmed below for the more commonly used CMAC structures
            // for the HEUAV application (n<=4 and L<=20 or 25 and high quality).
                    iomap[i].offset[j][k]=float(j+1)*setup.input_resolution[k];
                    // n=2
                    if(setup.in_dimensions==2 && iomap[i].levels==5 && k==1)
                    {
                        iomap[i].offset[j][1]=float(j+1)*2*setup.input_resolution[1];
                    }
                    if(setup.in_dimensions==2 && iomap[i].levels==8 && k==1)
```

```
{
    iomap[i].offset[j][1]=float(j+1)*3*setup.input_resolution[1];
}
if(setup.in_dimensions==2 && iomap[i].levels==10 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*3*setup.input_resolution[1];
}
if(setup.in_dimensions==2 && iomap[i].levels==13 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*5*setup.input_resolution[1];
}
if(setup.in_dimensions==2 && iomap[i].levels==15 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*4*setup.input_resolution[1];
}
if(setup.in_dimensions==2 && iomap[i].levels==17 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*4*setup.input_resolution[1];
}
if(setup.in_dimensions==2 && iomap[i].levels==18 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*5*setup.input_resolution[1];
}
// n=3
if(setup.in_dimensions==3 && iomap[i].levels==5 && k==2)
{
    iomap[i].offset[j][2]=float(j+1)*2*setup.input_resolution[2];
}
if(setup.in_dimensions==3 && iomap[i].levels==7 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*2*setup.input_resolution[1];
}
if(setup.in_dimensions==3 && iomap[i].levels==7 && k==2)
{
    iomap[i].offset[j][2]=float(j+1)*3*setup.input_resolution[2];
}
if(setup.in_dimensions==3 && iomap[i].levels==9 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*2*setup.input_resolution[1];
}
if(setup.in_dimensions==3 && iomap[i].levels==9 && k==2)
{
    iomap[i].offset[j][2]=float(j+1)*4*setup.input_resolution[2];
}
if(setup.in_dimensions==3 && iomap[i].levels==14 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*3*setup.input_resolution[1];
}
if(setup.in_dimensions==3 && iomap[i].levels==14 && k==2)
{
    iomap[i].offset[j][2]=float(j+1)*5*setup.input_resolution[2];
}
if(setup.in_dimensions==3 && iomap[i].levels==19 && k==1)
{
    iomap[i].offset[j][1]=float(j+1)*3*setup.input_resolution[1];
}
```

```
                    if(setup.in_dimensions==3 && iomap[i].levels==19 && k==2)
                    {
                        iomap[i].offset[j][2]=float(j+1)*7*setup.input_resolution[2];
                    }
                    if(setup.in_dimensions==3 && iomap[i].levels==25 && k==1)
                    {
                        iomap[i].offset[j][1]=float(j+1)*3*setup.input_resolution[1];
                    }
                    if(setup.in_dimensions==3 && iomap[i].levels==25 && k==2)
                    {
                        iomap[i].offset[j][2]=float(j+1)*8*setup.input_resolution[2];
                    }
                    // Check offset using modulus function
                    if(iomap[i].offset[j][k]>iomap[i].cell_width[j][k])
                    {
                    iomap[i].offset[j][k]=float(fmod(iomap[i].offset[j][k],iomap[i].cell_width[j][k]));
                    }
                }
                else
                {
                    iomap[i].offset[j][k]=iomap[i].offset_size;
                    iomap[i].offset[j][k]*=(float)(j)*(iomap[i].cell_width[j][k]);
                }
            } // input loop
        } // level loop
    } // output loop
    printf("\n\n");

    /*
    If the option to read in existing weights from an already trained CMAC architecture is set,
    then the read_wts() function is called; otherwise, all of the CMAC weights are initialized
    to 0.0.
    */

    if(setup.read_in_wts==0)
    {
        for(i=0;i<setup.out_dimensions;++i)
        {
            for(j=0;j<iomap[i].levels;++j)
            {
                for(k=0;k<iomap[i].hash_size[j];++k)
                {
                    iomap[i].working_memory[j][k]=0.0;
                }
            }
        }
    }
    else
    {
        read_wts();
    }

    return *this;
}

/*
```

This function computes the hashed address containing the CMAC weight values in the working memory array.

In the class implementation, let n be the dimension of the input space and a_i and b_i denote the initial and final value, respectively, of the focus interval for input i. The following auxiliary variables are then defined:

offset=O: the offset for output component k, level j, and input i, given by
iomap[k].offset[j][i]
width=W: the cell width for output component k, level j, and input i, given by
iomap[k].cell_width[j][i]
init=I: the a_i value for the focus interval for input i, given by setup.min_x[i]
final=F: the b_i value for the focus interval for input i, given by setup.max_x[i]

For a given output component k, level j and input i, the first cell covers the interval (-infinity, I + O) or (-infinity, I + W) depending on whether the offset is nonzero. Hence, if the ith component of x, x_i, satisfies x_i<I, it is covered by the first cell.
Similarly, the last cell is interpreted as covering out to +infinity, so if x_i>F, it is covered by the last cell. OFFSET_ZERO denotes the value under which the offset is treated as zero. Two cases are described in detail in the original documentation.

This code has built-in diagnostic and check code which checks to see if the active cell calculations are correct. For speed, these diagnostics are typically turned off. If the value of the #defined variable check is set to 1, the built-in checking will be activated. This will require a recompile of the CMAC class.

```
*/

CMAC& CMAC::virtual_memory(int k,float *x)
{
    int i,j,previous_cells;
    float offset,width,init,final;
    float down,up;

    // routine finds the virtual memory address for output component k that corresponds to
    // input vector x.
    previous_cells=0;
    for(j=0;j<iomap[k].levels;++j)
    {
        // level loop
        for(i=0;i<setup.in_dimensions;++i)
        {
            // input loop
            offset=iomap[k].offset[j][i];
            width=iomap[k].cell_width[j][i];
            init=setup.min_x[i];
            final=setup.max_x[i];

            if(offset>OFFSET_ZERO)
            {
                if(x[i]<(init+offset)) iomap[k].active_cells[i]=0;
                else iomap[k].active_cells[i]=(int)(ceil((x[i]-init-offset)/width));
            }
            else iomap[k].active_cells[i]=(int)(floor((x[i]-init)/width));

            if(check==1)
            {
```

```
/*
Check active cells (subinterval) calculation:
1. O>OFFSET_ZERO ==> s=0  I<=x<I+O
                     s>0  I+O+(s-1)W<=x<I+O+sW
2. O<=OFFSET_ZERO ==> I+sW<=x<I+(s+1)W
*/
if(offset>OFFSET_ZERO)
{
    if(iomap[k].active_cells[i]==0)
    {
        if(!(x[i]<init+offset+1.0e-6))
        {
            printf("x[%d]=%12.8e\n",i,x[i]);
            printf("offset=%12.8e\n",offset);
            printf("init=%12.8e\n",init);
            printf("final=%12.8e\n",final);
            printf("width=%12.8e\n",width);
            printf("num_intervals[level=%3d][input=%3d]=%3d\n",j,i,
                iomap[k].num_intervals[j][i]);
            printf("active_cells[%3d]=%3d\n",i,iomap[k].active_cells[i]);
            printf("[%12.8e,%12.8e,%12.8e]\n",init,x[i],init+offset);
            exit(1);
        }
    }
    else
    {
        down=offset+(iomap[k].active_cells[i]-1)*width;
        up=down+width;
        if(!((init+down-1.0e-6)<=x[i] && x[i]<(init+up+1.0e-6)))
        {
            printf("x[%d]=%12.8e\n",i,x[i]);
            printf("offset=%12.8e\n",offset);
            printf("init=%12.8e\n",init);
            printf("final=%12.8e\n",final);
            printf("width=%12.8e\n",width);
            printf("num_intervals[level=%3d][input=%3d]=%3d\n",j,i,
                iomap[k].num_intervals[j][i]);
            printf("active_cells[%3d]=%3d\n",i,iomap[k].active_cells[i]);
            printf("[%12.8e,%12.8e,%12.8e]\n",
            init+offset+(iomap[k].active_cells[i]-1)*width,x[i],
            init+offset+iomap[k].active_cells[i]*width);
            exit(1);
        }
    }
} // offset<OFFSET_ZERO
else
{
    down=offset+iomap[k].active_cells[i]*width;
    up=down+width;
    if(iomap[k].active_cells[i]==0)
    {
        if(!(x[i]<init+up+1.0e-6))
        {
            printf("x[%d]=%12.8e\n",i,x[i]);
            printf("offset=%12.8e\n",offset);
            printf("init=%12.8e\n",init);
```

```
                                    printf("final=%12.8e\n",final);
                                    printf("width=%12.8e\n",width);
                                    printf("num_intervals[level=%3d][input=%3d]=%3d\n",
                                        j,i,iomap[k].num_intervals[j][i]);
                                    printf("active_cells[%3d]=%3d\n",i,iomap[k].active_cells[i]);
                                    printf("[%12.8e,%12.8e,%12.8e]\n",
                                        init+offset+iomap[k].active_cells[i]*width,x[i],
                                        init+offset+(iomap[k].active_cells[i]+1)*width);
                                    exit(1);
                                }
                            }
                            else
                            {
                                if(!(init+down-1.0e-6<=x[i] && x[i]<init+up+1.0e-6))
                                {
                                    printf("x[%d]=%12.8e\n",i,x[i]);
                                    printf("offset=%12.8e\n",offset);
                                    printf("init=%12.8e\n",init);
                                    printf("final=%12.8e\n",final);
                                    printf("width=%12.8e\n",width);
                                    printf("num_intervals[level=%3d][input=%3d]=%3d\n",j,i,
                                        iomap[k].num_intervals[j][i]);
                                    printf("active_cells[%3d]=%3d\n",i,iomap[k].active_cells[i]);
                                    printf("[%12.8e,%12.8e,%12.8e]\n",
                                        init+offset+iomap[k].active_cells[i]*width,x[i],
                                        init+offset+(iomap[k].active_cells[i]+1)*width);
                                    exit(1);
                                }
                            }
                        }
                    } // check active_cells calculations
                    if(diagnostic==1) printf("active_cells[%2d,%2d]=%3d\n",i,j,iomap[k].active_cells[i]);
                    if(iomap[k].active_cells[i]<0)
                    {
                        printf("BIG ERROR--NEGATIVE (ACTIVE CELLS) SUBINTERVAL
CALCULATION\n");
                        exit(1);
                    }
                } // input loop
                // routine finds the hashed memory address for output component k that corresponds to
              // input vector x for level j.

            iomap[k].working_address[j]=iomap[k].hash_function[j](setup.in_dimensions,iomap[k].active_cells,
                                            iomap[k].num_intervals[j],iomap[k].hash_size[j]);
                if(diagnostic==1) printf("Working address [%d]=%d\n",j,iomap[k].working_address[j]);
                if(A_MATRIX==1)
                {
                    iomap[k].A_vector[j]=previous_cells+iomap[k].working_address[j];
                    previous_cells=previous_cells+iomap[k].num_cells[j];
                    if(diagnostic==1) printf("Association vector=%d\n",iomap[k].A_vector[j]);
                }
            } // level loop for kth iomap

            return *this;
}
```

```
// read and store training data in vectors in and out

CMAC& CMAC::read_training_data(float ***in,float ***out)
{
    int i,j,in_size,out_size,set_size;
    float *min,*max,diff;
    FILE *fd;

    in_size=setup.in_dimensions;
    out_size=setup.out_dimensions;
    set_size=setup.training_set_size;

    // open file and read training data
    if((fd=fopen(setup.training_file,"r"))==NULL)
    {
        printf("\nCan't open training file %s\n",setup.training_file);
        exit(0);
    }

    get_2dmatrix_float(in,set_size+1,in_size);
    get_2dmatrix_float(out,set_size+1,out_size);

    for(j=0;j<set_size;++j)
    {
        // read and store ith component of jth input vector
        for(i=0;i<in_size;++i) fscanf(fd,"%f",&(*in)[j][i]);
        // read and store ith component of jth desired output vector
        for(i=0;i<out_size;++i) fscanf(fd,"%f",&(*out)[j][i]);
    }
    fclose(fd);

    min=new float[setup.in_dimensions];
    max=new float[setup.in_dimensions];

    // find each input coordinate's interval
    for(i=0;i<in_size;++i)
    {
        max[i]=NEGATIVE_INFINITY;
        min[i]=INFINITY;
        for(j=0;j<set_size;++j)
        {
            if(max[i]<(*in)[j][i]) max[i]=(*in)[j][i];
            if(min[i]>(*in)[j][i]) min[i]=(*in)[j][i];
        }
    }

    // adjust endpoints of interval if autosize is desired
    for(i=0;i<in_size;++i)
    {
        if(setup.autosize==1)
        {
            setup.min_x[i]=min[i]-setup.inflate_bottom*(max[i]-min[i]);
            setup.max_x[i]=max[i]+setup.inflate_top*(max[i]-min[i]);
        }
        else
        {
                setup.min_x[i]=setup.common_start;
                setup.max_x[i]=setup.common_end;
            }
            printf("Input widths (min/max, training data):  Input[%d]=%12.6f/%12.6f\n",i,
                    setup.min_x[i],setup.max_x[i]);
        }

        // determine input resolution
        for(i=0;i<in_size;++i)
        {
            diff=0;
            for(j=1;j<set_size;++j)
            {
                if(((*in)[j][i]-(*in)[j-1][i])>0) diff=(*in)[j][i]-(*in)[j-1][i];
            }
            setup.input_resolution[i]=diff;
            printf("Input resolution:  Input[%d]=%12.6f\n",i,setup.input_resolution[i]);
        } // input loop

        delete min;
        delete max;

        return *this;
}

// read and store testing data in vectors in and out

CMAC& CMAC::read_testing_data(float ***in,float ***out)
{
        int i,j,in_size,out_size,set_size;
        float *min,*max;
        FILE *fd;

        in_size=setup.in_dimensions;
        out_size=setup.out_dimensions;
        set_size=setup.testing_set_size;

        // open file and read testing data
        if((fd=fopen(setup.testing_file,"r"))==NULL)
        {
            printf("\nCan't open testing file %s\n",setup.testing_file);
            exit(0);
        }

        get_2dmatrix_float(in,set_size+1,in_size);
        get_2dmatrix_float(out,set_size+1,out_size);

        for(j=0;j<set_size;++j)
        {
            // read and store ith component of jth input vector
            for(i=0;i<in_size;++i) fscanf(fd,"%f",&(*in)[j][i]);
            // read and store ith component of jth desired output vector
            for(i=0;i<out_size;++i) fscanf(fd,"%f",&(*out)[j][i]);
        }
        fclose(fd);
        min=new float[setup.in_dimensions];
        max=new float[setup.in_dimensions];

        // find each input coordinate's interval
        for(i=0;i<setup.in_dimensions;++i)
        {
            max[i]=NEGATIVE_INFINITY;
            min[i]=INFINITY;
            for(j=0;j<setup.testing_set_size;++j)
            {
                if(max[i]<(*in)[j][i]) max[i]=(*in)[j][i];
                if(min[i]>(*in)[j][i]) min[i]=(*in)[j][i];
            }
            printf("Input widths (min/max, test data):  Input[%d]=%12.6f/%12.6f\n",i,min[i],max[i]);

            if(min[i]<setup.min_x[i] || max[i]>setup.max_x[i])
            {
                printf("\nError!\n");
                printf("[minTrain,minTest,maxTrain,maxTest][%d]=[%10.6f,%10.6f,%10.6f,%10.6f]\n",i,
                    setup.min_x[i],min[i],setup.max_x[i],max[i]);
            }
        }
        delete min;
        delete max;

        return *this;
    }

    // print out weights both to console (if desired) and to file name_weights.txt

    #define write_to_console (0)
    CMAC& CMAC::write_wts(void)
    {
        FILE *fd;
        int i,j,k;

        // open file to write weights to
        if((fd=fopen(setup.wts_file,"w"))==NULL)
        {
            printf("\nCan't open file %s\n",setup.wts_file);
            exit(0);
        }

        // write data to file and console, if desired
        for(i=0;i<setup.out_dimensions;++i)
        {
            if(write_to_console==1) printf("\nListing of iomap[%d] weights:\n",i);
            for(j=0;j<iomap[i].levels;++j)
            {
                if(write_to_console==1) printf("\nlevel %3d\n",j);
                for(k=0;k<iomap[i].hash_size[j];++k)
                {
                    if(write_to_console==1)
                        printf("working_memory[level=%d][element=%d]=%12.6f\n",
                            j,k,iomap[i].working_memory[j][k]);
                    fprintf(fd,"%14.10f ",iomap[i].working_memory[j][k]);
                if((k+1)%4==0)
                {
                    if(write_to_console==1) printf("\n");
                    fprintf(fd,"\n");
                }
            } // cells loop
            if(write_to_console==1) printf("\n");
            fprintf(fd,"\n");
        } // levels loop
    } // output dimensions loop

    fclose(fd);
    return *this;
}


// read in weights from file and output to console (if desired)

CMAC& CMAC::read_wts(void)
{
    FILE *fd;
    int i,j,k;

    // open file
    if((fd=fopen(setup.wts_init_file,"r"))==NULL)
    {
        printf("\nCan't open input file %s\n",setup.wts_init_file);
        exit(0);
    }

    // read data from file and write to console, if desired
    for(i=0;i<setup.out_dimensions;++i)
    {
        if(write_to_console==1) printf("\n iomap[%d] weights \n",i);
        for(j=0;j<iomap[i].levels;++j)
        {
            if(write_to_console==1) printf("\nLevel %3d\n",j);
            for(k=0;k<iomap[i].hash_size[j];++k)
            {
                fscanf(fd,"%f\n",&(iomap[i].working_memory[j][k]));
                if(write_to_console==1) printf("working_memory[level=%d][element=%d]=%12.6f\n",
                    j,k,iomap[i].working_memory[j][k]);
            } // cells loop
        } // levels loop
    } //output dimensions loop

    fclose(fd);
    return *this;
}

/*
This function echoes to the screen the critical parameters of the CMAC architecture.  Since
there can be many such parameters for each choice of output dimension, this code defaults to
printing the required data for output component 0.  This choice can be changed by altering the
#defined variable WHICH_OUTPUT_COMPONENT to whichever index is needed.  This, of course,
implies a recompile of the class.
```

```
*/

#define WHICH_OUTPUT_COMPONENT (0)
CMAC& CMAC::echo(void)

{
    int total_flattened_cells,i,j,k,m,n;
    float temp;

    // print configuration data to the screen
    printf("\n\n\n");
    printf("The data below is an echoing of the data stored in the instantiated CMAC object:");
    printf("setup.config_file=%s\n",setup.config_file);
    printf("setup.structure_file=%s\n",setup.structure_file);
    printf("setup.training_file=%s\n",setup.training_file);
    printf("setup.testing_file=%s\n",setup.testing_file);
    printf("setup.wts_init_file=%s\n",setup.wts_init_file);
    printf("setup.wts_file=%s\n",setup.wts_file);
    printf("setup.training_set_size=%3d\n",setup.training_set_size);
    printf("setup.testing_set_size=%2d\n",setup.testing_set_size);
    printf("setup.learning_rate=%8.6f\n",setup.learning_rate);
    printf("setup.in_dimensions=%1d\n",setup.in_dimensions);
    printf("setup.out_dimensions=%1d\n",setup.out_dimensions);

    for(i=0;i<setup.in_dimensions;++i)
    {
        printf("input interval[%1d]=[%12.6f,%12.6f]\n",i,setup.min_x[i],setup.max_x[i]);
    }
    printf("\n");

    // print data for the CMAC structure of one output dimension
    for(i=0;i<setup.out_dimensions;++i)
    {
        printf("\n");
        printf("\n");
        printf("CMAC output [%1d] parameters:\n",WHICH_OUTPUT_COMPONENT);
        printf("setup.file[%1d]=%s\n",i,setup.file[i]);
        printf("number levels=%2d\n",iomap[WHICH_OUTPUT_COMPONENT].levels);
        printf("hash=%2d\n",iomap[WHICH_OUTPUT_COMPONENT].hash_size_construct);
        printf("LEVEL  INPUT    OFFSET     CELL WIDTH  HASH_SIZE\n");
        for(j=0;j<iomap[WHICH_OUTPUT_COMPONENT].levels;++j)
        {
            for(k=0;k<setup.in_dimensions;++k)
            {
            printf("%3d    %3d  %12.6f %12.6f     %3d\n",j,k,
                iomap[WHICH_OUTPUT_COMPONENT].offset[j][k],
                iomap[WHICH_OUTPUT_COMPONENT].cell_width[j][k],
                iomap[WHICH_OUTPUT_COMPONENT].hash_size[j]);
            }
        }
    }
    printf("\n");

    /*
    This block of code computes the total number of cells that the CMAC architecture would
    have if the level structure would be collapsed or ``flattened" from L levels to one level.
    Essentially, if there are O output components, then for output component i, there are L_i
    levels in the architecture with a fixed cell width W_i and an interval [a_i,b_i]
    where the CMAC cells are active.  Thus, the number of cells on a given level is given by
    S_i=(b_i-a_i)/W_i and the total number of cells for the output component i is T_i=S_ixL_i.
    This number must be converted to the nearest integer--call this integer M_i.  The total
    number of flattened cells for the full CMAC architecture is then the sum of M_i from 0 to
    (O-1).
    */

    total_flattened_cells=0;
    printf("\n");
    for(i=0;i<setup.out_dimensions;++i)
    {
        n=1;
        for(k=0;k<setup.in_dimensions;++k)
        {
            temp=(setup.max_x[k]-setup.min_x[k])*iomap[i].levels/iomap[i].cell_width[1][k];
            m=(int)temp;
            if((temp-m)>0.0) m+=1;
            printf("flattened cells[output=%d, levels=%d][%d input]=%d\n",i,iomap[i].levels,k,m);
            n*=m;
        }
        total_flattened_cells+=n;
        printf("Total flattened cells estimate for %d output=%d\n",setup.out_dimensions,
            total_flattened_cells);
        printf("Memory savings using training set size (CMAC vs. look-up table, use!!)=%f\n",
            setup.training_set_size/float(iomap[i].total_cells));
        printf("Memory savings using flattened cells estimate (CMAC vs. look-up table)=%f\n",
            float(total_flattened_cells)/float(iomap[i].total_cells));
    }

    return *this;
}

// The following functions used to obtain various data are self-explanatory.

CMAC& CMAC::get_input_size(int *input_size)
{
    *input_size=setup.in_dimensions;
    return *this;
}

CMAC& CMAC::get_output_size(int *output_size)
{
    *output_size=setup.out_dimensions;
    return *this;
}

CMAC& CMAC::get_train_size(int *train_size)
{
    *train_size=setup.training_set_size;
    return *this;
}

CMAC& CMAC::get_test_size(int *test_size)
{
    *test_size=setup.testing_set_size;
    return *this;
}

CMAC& CMAC::get_levels(int which_output,int *levels)
{
    *levels=iomap[which_output].levels;
    return *this;
}

CMAC& CMAC::get_number_intervals(int which_output,int which_level,int which_input,int *number)
{
    *number=iomap[which_output].num_intervals[which_level][which_input];
    return *this;
}

CMAC& CMAC::get_total_cells(int which_output,int *total_cells)
{
    *total_cells=iomap[which_output].total_cells;
    return *this;
}

CMAC& CMAC::get_A_value(int which_output,int which_level,int *value)
{
    *value=iomap[which_output].A_vector[which_level];
    return *this;
}

CMAC& CMAC::get_min_x(int which_input,float *min)
{
    *min=setup.min_x[which_input];
    return *this;
}

CMAC& CMAC::get_max_x(int which_input,float *max)
{
    *max=setup.max_x[which_input];
    return *this;
}

CMAC& CMAC::get_width(int which_output,int which_level,int which_input,float *width)
{
    *width=iomap[which_output].cell_width[which_level][which_input];
    return *this;
}

CMAC& CMAC::get_working_address(int which_output,int *address)
{
    int i;
    for(i=0;i<iomap[which_output].levels;++i) address[i]=iomap[which_output].working_address[i];
    return *this;
}

CMAC& CMAC::load_input_size(int input_size)
{
    setup.in_dimensions=input_size;
    return *this;
}

CMAC& CMAC::load_output_size(int output_size)
{
    setup.out_dimensions=output_size;
    return *this;
}

CMAC& CMAC::load_train_size(int train_size)
{
    setup.training_set_size=train_size;
    return *this;
}

CMAC& CMAC::load_test_size(int test_size)
{
    setup.testing_set_size=test_size;
    return *this;
}

CMAC& CMAC::number(void)
{
    int i,j,k,p,n,nmax;
    float offset,width,init,final,temp;

    /*-------------------------------------------------------------
    Case 0:  no degeneracies:  (offset>OFFSET_ZERO)
    I----------I+O--------I+O+W------I+O+2W-----/ /----I+O+NW-----F
    subintervals:
    0       1        2              N       N+1

    Case 1:  degeneracies:  (offset<=OFFSET_ZERO)
    I=I+O--------I+W--------I+2W----------------/ /---I+MW--------F
    subintervals:
     0       1        2             M-1       M
    */

    for(k=0;k<setup.out_dimensions;++k)
    {
        for(j=0;j<iomap[k].levels;++j)
        {
            for(i=0;i<setup.in_dimensions;++i)
            {
                offset=iomap[k].offset[j][i];
                width=iomap[k].cell_width[j][i];
                init=setup.min_x[i];
                final=setup.max_x[i];
                temp=(final-init-offset)/width;
                if(temp<0.0)
                {
                    printf("iomap[%d].num_intervals calculation negative\n",k);
                    printf("final=%12.6e init=%12.6e offset=%12.6e\n",final,init,offset);
                }
                iomap[k].num_intervals[j][i]=(int)temp;
                if((temp-(int)temp-OFFSET_ZERO)>0) iomap[k].num_intervals[j][i]+=1;
                if(offset>OFFSET_ZERO) iomap[k].num_intervals[j][i]+=1;
                p=iomap[k].num_intervals[j][i];
                if(check==1)
                {
                    if(offset>OFFSET_ZERO)
                    {
                        if(!(init+offset+(p-2)*width-OFFSET_ZERO<=final&&final<=init+offset+
                            (p-1)*width+OFFSET_ZERO))
                        {
                            printf("init=%12.8e\n",init);
                            printf("final=%12.8e\n",final);
                            printf("offset=%12.8e\n",offset);
                            printf("width=%12.8e\n",width);
                            printf("(F-I-O)/W=%12.8e\n",temp);
                            printf("(int)temp=%3d\n",(int)temp);
                            printf("ceil(temp)=%12.6e\n",ceil(temp));
                            printf("floor(temp)=%12.6e\n",floor(temp));
                            printf("iomap[%d].num_intervals[level=%3d][input=%3d]=%3d\n",k,j,i,p);
                            printf("[%12.8e,%12.8e,%12.8e]\n",init+offset+(p-2)*width,final,init+
                                offset+(p-1)*width);
                            exit(1);
                        }
                    }
                    else
                    {
                        if(!(init+offset+(p-1)*width-OFFSET_ZERO<=final&&final<=init+offset+(p)*
                            width+OFFSET_ZERO))
                        {
                            printf("init=%12.8e\n",init);
                            printf("offset=%12.8e\n",offset);
                            printf("width=%12.8e\n",width);
                            printf("iomap[%d].num_intervals[level=%3d][input=%3d]=%3d\n",k,j,i,p);
                            printf("[%12.8e,%12.8e,%12.8e]\n",init+offset+(p-1)*width,final,init+
                                offset+(p)*width);
                            exit(1);
                        }
                    }
                } // check iomap[k].number calculation for level j, input component i
                if(check==1)
                    printf("iomap[%d].num_intervals[level=%3d][input=%3d]=%3d\n",k,j,i,p);
            } // input loop
        } // level loop
    } // output component loop

    // calculate number of cells per level and total cells
    for(i=0;i<setup.out_dimensions;++i)
    {
        nmax=0;
        iomap[i].total_cells=0;
        for(j=0;j<iomap[i].levels;++j)
        {
            n=1;
            for(k=0;k<setup.in_dimensions;++k)
            {
                printf("Cells for one input[output=%d][level=%d][input=%d]=%d\n",i,j,k,
                    iomap[i].num_intervals[j][k]);
                    n*=iomap[i].num_intervals[j][k];
                    if(n>nmax) nmax=n;
                }
                iomap[i].num_cells[j]=n;
                printf("Number of cells in layer %d = %d.\n",j,iomap[i].num_cells[j]);
                iomap[i].total_cells=iomap[i].total_cells+iomap[i].num_cells[j];
            }
            printf("Total number of cells (memory requirement)=%d \n",iomap[i].total_cells);
            printf("Max number of cells in a level (minimum hash size for no hashing)=%d\n",nmax);
        }
        return *this;
    }

    float CMAC::compute_rms(int sample_size,float **in,float **out)
    {
        int i,k;
        float rms, big;

        rms=0.0;
        big=-1.0;
        for(k=0;k<sample_size;++k)
        {
            cmac_eval(1,in[k]);
            if(diagnostic==1)
            {
                for(i=0;i<setup.in_dimensions;++i)
                    printf("in[%3d][%3d]=%12.6f\n",k,i,in[k][i]);
            }
            for(i=0;i<setup.out_dimensions;++i)
            {
                rms+=(out[k][i]-iomap[i].output)*(out[k][i]-iomap[i].output);
                if(float(fabs(out[k][i]-iomap[i].output))>big)
                    big=float(fabs(out[k][i]-iomap[i].output));
            } // output dimension loop
        } // sample loop
        rms=float(sqrt(rms/(float)sample_size/(float)setup.out_dimensions));
        //printf("Largest absolute error:  %12.6f\n",big);
        return(rms);
    }

/*
This function evaluates the output of the CMAC architecture for a given input x.  Note that if
the working_memory structure holds current addressing information, the value of 0 for the
argument do_address can be passed in and the address calculation function virtmem() will not be
called.  If address information is required, setting the argument do_address to 1 enables the
address calculation call to virtmem().
*/

float *CMAC::cmac_eval(int do_address,float *x)
{
    int i,p,base;
    static float *value;

    value=new float[setup.out_dimensions];

    if(diagnostic==1)
    {
        for(i=0;i<setup.in_dimensions;++i) printf("x[%d]=%12.6f\n",i,x[i]);
    }
    for(i=0;i<setup.out_dimensions;++i)
    {
        if(do_address==1) virtual_memory(i,x);
        iomap[i].output=0.0;
        for(p=0;p<iomap[i].levels;++p)
        {
            base=iomap[i].working_address[p];
            iomap[i].output+=iomap[i].working_memory[p][base];
        } // level loop
        value[i]=iomap[i].output;
    } // output dimension loop
    if(diagnostic==1)
    {
        for(i=0;i<setup.out_dimensions;++i) printf("iomap[%d].output=%5.1f\n",i,iomap[i].output);
    }
    return(value);
}

/*
The training algorithm uses the delta rule.  The training algorithm is computationally
efficient, fast, and local.
*/

CMAC& CMAC::train(int do_address,int sample_size,float **in,float **out)
{
    int i,j,k,base;

    for(k=0;k<sample_size;++k)
    {
        // evaluate address and compute cmac output
        cmac_eval(1,in[k]);
        for(i=0;i<setup.out_dimensions;++i)
        {
            for(j=0;j<iomap[i].levels;++j)
            {
                base=iomap[i].working_address[j];
                iomap[i].working_memory[j][base]
                    +=setup.learning_rate*(out[k][i]-iomap[i].output)/(float)iomap[i].levels;
            } // levels loop
        } // output component loop
    } // sample loop

    return *this;
}
```

BLANK

Bibliography

[1] C. C. Chan and K. T. Chau, *Modern Electric Vehicle Technology*. Oxford, New York: Oxford University Press, 2001.

[2] E. C. Aldridge and J. P. Stenbit, "Unmanned Aerial Vehicles Roadmap: 2002-2027," Office of the Secretary of Defense, 2003.

[3] C. Perazzola, "Tomorrow's Ground Power Today," presented at SAE June TOPTEC Conference, 2002.

[4] J. R. Wilson, "UAVs: A Worldwide Roundup," in *Aerospace America*, 2003, pp. 30-35.

[5] S. B. Wilson, "Micro Air Vehicle Project," Defense Advanced Research Projects Agency.

[6] A. Shalal-Esa, "Pentagon Gives Boeing UAV Contract," Reuters, 2003.

[7] J. Conrow, "The Helios Comes Down to Earth-A Winner," AeroVironment, 2001.

[8] J. J. Berton, J. E. Freeh, and T. J. Wickenheiser, "An Analytical Performance Assessment of a Fuel Cell-Powered Small Electric Airplane," presented at Symposium on Novel and Emerging Vehicle and Vehicle Technology Concepts, Brussels, Belgium, 2003.

[9] J. E. Freeh, A. Liang, J. J. Berton, and T. J. Wickenheiser, "Electrical Systems Analysis at NASA Glenn Research Center: Status and Prospects," presented at Symposium on Novel and Emerging Vehicle and Vehicle Technology Concepts, Brussels, Belgium, 2003.

[10] I. Husain, *Electric and Hybrid Vehicles Design Fundamentals*. Boca Rotan, FL: CRC Press, 2003.

[11] M. C. Pera, D. Hissel, and J. M. Kauffman, "Fuel Cell Systems for Electrical Vehicles: An Overview," in *IEEE Vehicular Technology Society News*, vol. 49, 2002, pp. 9-14.

[12] B. Johnston, et al., "The Continued Design and Development of the University of California-Davis FutureCar," *SAE Paper 980487*, 1998.

[13] A. F. Burke, "Hybrid/Electric Vehicle Design Options and Evaluations," *SAE Document SP-915*, pp. 53-77, 1992.

[14] C. Kim, E. NamGoong, S. Lee, T. Kim, and H. Kim, "Fuel Economy Optimization for Parallel Hybrid Vehicles with CVT," *SAE Paper 1999-01-1148*, 1999.

[15] B. Kleback, S. Inman, and R. Noss, "Design and Development of the 2002 Penn State University Parallel Hybrid Electric Explorer, the Wattmuncher," *SAE Paper 2003-01-1258*, 2003.

[16] N. Meyr, et al., "Design and Development of the 2002 UC Davis FutureTruck," *SAE Paper 2003-01-1263*, 2003.

[17] C. Bond, et al., "Design and Development of the 2003 University of Alberta Hybrid Electric Vehicle," *SAE Paper 2003-01-1268*, 2003.

[18] Honda, "2002 Honda Insight," American Honda Motor Company, Inc., 2001.

[19] Toyota, "Toyota Hybrid and Electric Vehicles," International Public Affairs Department, 2000.

[20] I. Matsuo, S. Nakazawa, H. Maeda, and E. Inada, "Development of a High-Performance Hybrid Propulsion System Incorporating a CVT," *SAE Paper 2000-01-0992*, pp. 1-9, 2000.

[21] S. Sasaki, "Toyota's Newly Developed Powertrain," presented at International Symposium on Power Semiconductor Devices and ICs, Kyoto, Japan, 1998.
[22] M. Harmats and D. Weihs, "Hybrid-Propulsion High-Altitude Long-Endurance Remotely Piloted Vehicle," *Journal of Aircraft*, vol. 36, pp. 321-331, 1999.
[23] M. Alexander, et al., "Design and Development of the 2000 UC Davis FutureTruck," *SAE Document SP-1617*, 2001.
[24] N. Meyr, et al., "Design and Development of the 2001 UC Davis FutureTruck," *SAE Paper 2001-01-1210*, 2002.
[25] "First Place Engines, 1.3 Cubic Inch Gas Engine," First Place Engines, 2004.
[26] D. Palombo and G. D. Miller, "A High Efficiency Electric Motor Propeller Propulsion System for Solar Powered UAVs," *SAE Paper 981263*, pp. 127-131, 1998.
[27] R. W. Schurhoff, "M.S. Thesis: The Development and Evaluation of an Optimal Powertrain Control Strategy for a Hybrid Electric Vehicle," in *Dept. of Mechanical and Aeronautical Engineering*. Davis, CA: University of California-Davis, 2002.
[28] P. Drozdz and A. Zettel, "Method and Apparatus for Adaptive Hybrid Vehicle Control," in *U.S. Patent and Trademark Office Database*. United States: Azure Dynamics, Inc., 2001.
[29] F. H. Glanz, T. W. Miller, and L. G. Kraft, "An Overview of the CMAC Neural Network," presented at IEEE Conference on Neural Networks for Ocean Engineering, 1991.
[30] W. T. Miller, F. H. Glanz, and L. G. Kraft, "CMAC: An Associative Neural Network Alternative to Backpropagation," *Proceedings of the IEEE*, vol. 78, pp. 1561-1567, 1990.
[31] H. Shiraishi, S. L. Ipri, and D. D. Cho, "CMAC Neural Network Controller for Fuel-Injection Systems," *IEEE Transactions on Control Systems Technology*, vol. 3, pp. 32-38, 1995.
[32] L. C. Iwan and R. F. Stengel, "The Application of Neural Networks to Fuel Processors for Fuel-Cell Vehicles," *IEEE Transactions on Vehicular Technology*, vol. 50, pp. 125-143, 2001.
[33] W. T. Miller, R. P. Hewes, F. H. Glanz, and L. G. Kraft, "Real-Time Dynamic Control of an Industrial Manipulator Using a Neural-Network-Based Learning Controller," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 1-9, 1990.
[34] L. G. Kraft and J. Pallota, "Real-Time Vibration Control Using CMAC Neural Networks with Weight Smoothing," presented at American Control Conference, Chicago, IL, 2000.
[35] L. Li and C. Hou, "The Study of Application of FCMAC Neural Network in the Industrial Process On-Line Identification and Control," presented at 9th International Conference on Neural Information Processing, 2002.
[36] A. L. Kun and T. W. Miller, "Unified Walking Control for a Biped Robot Using Neural Networks," presented at IEEE ISIC/CIRA/ISAS Joint Conference, Gaithersburg, MD, 1998.
[37] A. L. Kun and T. W. Miller, "Control of Variable-Speed Gaits for a Biped Robot," in *IEEE Robotics and Automation Magazine*, vol. 6: IEEE, 1999, pp. 19-29.

[38] W. T. Miller, "Real-Time Neural Network Control of a Biped Walking Robot," *IEEE Control Systems Magazine*, vol. 14, pp. 41-48, 1994.

[39] W. T. Miller and C. M. Aldrich, "Rapid Learning Using CMAC Neural Networks: Real Time Control of an Unstable System," presented at 5th International Symposium on Intelligent Control, 1990.

[40] J. Nelson and L. G. Kraft, "Real-Time Control of an Inverted Pendulum System Using Complementary Neural Network and Optimal Techniques," presented at American Control Conference, Baltimore, MD, 1994.

[41] L. G. Kraft and D. Dietz, "Time Optimal Control Using CMAC Neural Networks," presented at American Control Conference, Baltimore, MD, 1994.

[42] J. Nelson and L. G. Kraft, "Using CMAC Neural Networks and Optimal Control," presented at IEEE International Conference on Neural Networks, 1995.

[43] A. B. Francisco, "M.S. Thesis: Implementation of an Ideal Operating Line Control Strategy for Hybrid Electric Vehicles," in *Dept. of Mechanical and Aeronautical Engineering*. Davis, CA: University of California-Davis, 2002.

[44] V. H. Johnson, K. B. Wipke, and D. J. Rausen, "HEV Control Strategy for Real-Time Optimization of Fuel Economy and Emissions," *SAE Paper 2000-01-1543*, 2000.

[45] T. Mayer and D. Schroder, "Simulation and Hierarchical Controller Design for a Special Hybrid Drivetrain," presented at 6th European Conference on Power Electronics and Applications, 1995.

[46] J. J. E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[47] A. Piccolo and L. Ippolito, "Optimisation of Energy Flow Management in Hybrid Electric Vehicles via Genetic Algorithms," presented at IEEE/ASME International Conference on Advanced Intelligent Mechatronics Proceedings, Como, Italy, 2001.

[48] A. Kleimaier and D. Schroder, "An Approach for the Online Optimized Control of a Hybrid Powertrain," presented at 7th International Workshop on Advanced Motion Control, 2002.

[49] S. R. Cikankek, K. E. Bailey, R. C. Baraszu, and B. K. Powell, "Control System and Dynamic Model Validation for a Parallel Hybrid Electric Vehicle," presented at American Control Conference, San Diego, CA, 1999.

[50] S. R. Cikankek, K. E. Bailey, and B. K. Powell, "Parallel Hybrid Electric Vehicle Dynamic Model and Powertrain Control," presented at American Control Conference, Albuquerque, NM, 1997.

[51] B. K. Powell, K. E. Bailey, and S. R. Cikankek, "Dynamic Modeling and Control of Hybrid Electric Vehicle Powertrain Systems," in *Control Systems Magazine*, vol. 18, 1998, pp. 17-33.

[52] Y. Yang, M. Parten, J. Berg, and T. Maxwell, "Modeling and Control of a Hybrid Electric Vehicle," presented at IEEE Vehicular Technology Conference, 2000.

[53] X. He and J. W. Hodgson, "Modeling and Simulation for Hybrid Electric Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, pp. 235-243, 2002.

[54] P. Bowles, H. Peng, and X. Zhang, "Energy Management in a Parallel Hybrid Electric Vehicle with a Continuously Variable Transmission," presented at American Control Conference, Chicago, IL, 2000.

[55] T. C. Chang, "A Logic-Based Switch Torque Control for Parallel-Type Hybrid Electric Vehicle," *Journal of Chinese Society of Mechanical Engineers*, vol. 21, pp. 527-536, 2000.

[56] D. L. Buntin and J. W. Howze, "A Switching Logic Controller for a Hybrid Electric/ICE Vehicle," presented at American Control Conference, Seattle, WA, 1996.

[57] K. E. Bailey, S. R. Cikankek, and N. Sureshbabu, "Parallel Hybrid Electric Vehicle Torque Distribution Method," presented at American Control Conference, Anchorage, AK, 2002.

[58] N. Jalil, N. A. Kheir, and M. Salman, "A Rule-Based Energy Management Strategy for a Series Hybrid Vehicle," presented at American Control Conference, Albuquerque, NM, 1997.

[59] E. Biscarri, M. A. Tamor, and S. Murtuza, "Simulation of Hybrid Electric Vehicles with Emphasis on Fuel Economy Estimation," *SAE Paper 981132*, pp. 1-8, 1998.

[60] R. W. Schurhoff, "M.S. Thesis: The Development and Evaluation of an Optimal Powertrain Control Strategy for a Hybrid Electric Vehicle," in *Mechanical Engineering*. Davis, CA: University of California, 2002, pp. 149.

[61] A. A. Frank, "Charge Depletion Control Method and Apparatus for Hybrid Powered Vehicles," in *USPTO Patent Database*. USA: The Regents of the University of California, 1998.

[62] A. A. Frank, "Control Method and Apparatus for Internal Combustion Engine Electric Hybrid Vehicles," in *USPTO Patent Database*. USA: The Regents of the University of California, 2000.

[63] D. S. Naidu, *Optimal Control Systems*. Boca Raton, FL: CRC Press, 2003.

[64] F. L. Lewis, *Optimal Control*. New York: John Wiley & Sons, 1986.

[65] G. Paganelli, G. Ercole, A. Brahma, Y. Guezennec, and G. Rizzoni, "A General Formulation for the Instantaneous Control of the Power Split in Charge-Sustaining Hybrid Electric Vehicles," presented at 5th International Symposium on Advanced Vehicle Control, Ann Arbor, MI, 2000.

[66] G. Paganelli, M. Tateno, A. Brahma, G. Rizzoni, and Y. Guezennec, "Control Development for a Hybrid-Electric Sport-Utility Vehicle: Strategy, Implementation, and Field Test Results," presented at American Control Conference, Arlington, VA, 2001.

[67] S. Delprat, T. M. Guerra, G. Paganelli, J. Lauber, and M. Delhom, "Control Strategy Optimization for a Hybrid Parallel Powertrain," presented at American Control Conference, Arlington, VA, 2001.

[68] S. Delprat, T. M. Guerra, and J. Rimaux, "Control Strategies for Hybrid Vehicles: Optimal Control," presented at IEEE 56th Vehicular Technology Conference, 2002.

[69] G. Steinmauer and L. Re, "Optimal Control of Dual Power Sources," presented at IEEE International Conference on Control and Applications, Mexico City, Mexico, 2001.

[70] D. S. Bernstein, "Nonquadratic Cost and Nonlinear Feedback Control," *International Journal of Robust and Nonlinear Control*, vol. 3, pp. 211-229, 1993.

[71] S. E. Lyshevski, "Energy Conversion and Optimal Energy Management in Diesel-Electric Drivetrains of Hybrid-Electric Vehicles," *Energy Conversion and Management*, vol. 41, pp. 13-24, 2000.

[72] S. E. Lyshevski and C. Yokomoto, "Control of Hybrid-Electric Vehicles," presented at American Control Conference, Philadelphia, PA, 1998.

[73] S. Lyshevski, "Constrained Optimization and Control of Nonlinear Systems: New Results in Optimal Control," presented at Conference on Decision and Control, Kobe, Japan, 1996.

[74] S. E. Lyshevski, "Optimal Control of Nonlinear Continuous-Time Systems: Design of Bounded Controllers via Generalized Nonquadratic Functionals," presented at American Control Conference, Philadelphia, PA, 1998.

[75] R. Saeks, C. J. Cox, J. Neidhoefer, P. R. Mays, and J. J. Murray, "Adaptive Control of a Hybrid Electric Vehicle," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, pp. 213-234, 2002.

[76] R. Zhang and Y. Chen, "Control of Hybrid Dynamical Systems for Electric Vehicles," presented at American Control Conference, Arlington, VA, 2001.

[77] D. A. White and D. A. Sofge, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. New York: Van Nostrand Reinhold, 1992.

[78] U. Zoelch and D. Schroeder, "Dynamic Optimization Method for Design and Rating of the Components of a Hybrid Vehicle," *International Jounal of Vehicle Design*, vol. 19, pp. 1-13, 1998.

[79] A. Kleimaier and D. Schroder, "Optimization Strategy for Design and Control of a Hybrid Vehicle," presented at 6th International Workshop on Advanced Motion Control, 2000.

[80] R. Fellini, N. Michelena, P. Papalambros, and M. Sasena, "Optimal Design of Automotive Hybrid Powertrain Systems," presented at First International Symposium on Environmentally Conscious Design and Inverse Manufacturing, 1999.

[81] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[82] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton, N.J.,: Princeton University Press, 1962.

[83] C. C. Lin, J. M. Kang, J. W. Grizzle, and H. Peng, "Energy Management Strategy for a Parallel Hybrid Electric Truck," presented at American Control Conference, Arlington, VA, 2001.

[84] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Reading, MA: Addison-Wesley, 1995.

[85] A. Brahma, Y. Guezennec, and G. Rizzoni, "Dynamic Optimization of Mechanical/Electrical Power Flow in Parallel Hybrid Electric Vehicles," presented at 5th International Symposium on Advanced Vehicle Control, Ann Arbor, MI, 2000.

[86] A. Brahma, Y. Guezennec, and G. Rizzoni, "Optimal Energy Management in Series Hybrid Electric Vehicles," presented at American Control Conference, Chicago, IL, 2000.

[87] W. C. Morchin, "Energy Management in Hybrid Electric Vehicles," presented at Digital Avionics Systems Conference, 1998.

[88] H. Anton, *Calculus with Analytic Geometry*, 2nd ed. New York: John Wiley & Sons, Ltd., 1984.

[89] M. Zahran, A. Hanafy, O. Mahgoub, and M. Kamel, "FLC Based Photovoltaic Battery Diesel Hybrid System Management and Control," presented at 28th Photovoltaic Specialists Conference, 2000.

[90] J. Jantzen, "A Tutorial on Adaptive Fuzzy Control," www.eunite.org, 2002.

[91] S. Chiu, "Using Fuzzy Logic in Control Applications: Beyond Fuzzy PID Control," in *Control Systems Magazine*, vol. 18, 1998, pp. 100-104.

[92] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, pp. 3-11, 1965.

[93] L. A. Zadeh, "Fuzzy Logic," *IEEE Computer*, vol. 21, pp. 83-93, 1998.

[94] P. Singh, C. Fennie, and D. E. Reisner, "Logical Progression," in *Electric and Hybrid Vehicle Technology*, 2000, pp. 72-74.

[95] B. M. Baumann, G. Washington, B. C. Glenn, and G. Rizzoni, "Mechatronic Design and Control of Hybrid Electric Vehicles," *IEEE Transactions on Mechatronics*, vol. 5, pp. 58-72, 2000.

[96] R. E. King, *Computational Intelligence in Control Engineering*. New York: Marcel Dekker, Inc., 1999.

[97] L. X. Wang, "Design and Analysis of Fuzzy Identifiers of Nonlinear Dynamic Systems," *IEEE Transactions on Automatic Control*, vol. 40, pp. 11-23, 1995.

[98] M. Salman, N. J. Schouten, and N. A. Kheir, "Control Strategies for Parallel Hybrid Vehicles," presented at American Control Conference, Chicago, IL, 2000.

[99] N. J. Schouten, M. Salman, and N. A. Kheir, "Fuzzy Logic Control for Parallel Hybrid Vehicles," *IEEE Transactions on Control Systems Technology*, vol. 10, pp. 460-468, 2002.

[100] S. D. Farrall and R. P. Jones, "Energy Management in an Automotive Electric/Heat Engine Hybrid Powertrain Using Fuzzy Decision Making," presented at 1993 International Symposium on Intelligent Control, Chicago, IL, 1993.

[101] E. S. Koo, H. D. Lee, S. K. Sul, and J. S. Kim, "Torque Control Strategy for a Parallel Hybrid Vehicle Using Fuzzy Logic," presented at IEEE Industry Applications Conference, New York, 1998.

[102] H. D. Lee and S. K. Sul, "Fuzzy-Logic-Based Torque Control Strategy for Parallel-Type Hybrid Electric Vehicle," *IEEE Transactions on Industrial Electronics*, vol. 45, pp. 625-632, 1998.

[103] N. Schouten, "Fuzzy Logic Control for Parallel Hybrid Vehicles using PSAT," presented at Joint ADVISOR/PSAT Vehicle Systems Modeling User Conference, Southfield, MI, 2001.

[104] J. S. Won and R. Langari, "Fuzzy Torque Distribution Control for a Parallel-Hybrid Vehicle," presented at Joint ADVISOR/PSAT Vehicle Systems Modeling User Conference, Southfield, MI, 2001.

[105] E. Cerruto, A. Consoli, A. Raciti, and A. Testa, "Energy Flows Management in Hybrid Vehicles by Fuzzy Logic Controller," presented at Electrotechnical Conference, 1994.

[106] S. Sakai, S. Onimaru, M. Inagaki, and H. Asa, "Generator Control System for a Hybrid Vehicle Driven by an Electric Motor and an Internal Combustion Engine," in *U.S. Patent and Trademark Office Database*. United States: Nippon Soken, Inc., 1998.

[107] S. Ibaraki, "Control System for Hybrid Vehicle," in *U.S. Patent and Trademark Office Database*. United States: Honda Giken Kogyo Kabushiki Kaisha, 1999.

[108] C. P. Quigley, R. J. Ball, and R. P. Jones, "Fuzzy Modeling Approach to the Prediction of Journey Parameters for Hybrid Electric Vehicle Control," *Journal of Automobile Engineering*, vol. 214, pp. 875-885, 2000.

[109] C. Bourne, P. Faithfull, and C. Quigley, "Implementing Control of a Parallel Hybrid Vehicle," *International Journal of Vehicle Design*, vol. 17, pp. 649-662, 1996.

[110] C. P. Quigley, R. J. Ball, A. M. Vinsome, and R. P. Jones, "Predicting Journey Parameters for the Intelligent Control of a Hybrid Electric Vehicle," presented at IEEE International Symposium on Intelligent Control, Dearborn, MI, 1996.

[111] A. Brahma, B. Glenn, Y. Guezennec, T. Miller, G. Rizzoni, and G. Washington, "Modeling, Performance Analysis and Control Design of a Hybrid Sport-Utility Vehicle," presented at International Conference on Control Applications, Hawaii, 1999.

[112] A. Rajagopalan and G. Washington, "Intelligent Control of Hybrid Electric Vehicles Using GPS Information," *SAE Paper 2002-01-1936*, 2002.

[113] M. Agarwal, "A Systematic Classification of Neural-Network-Based Control," *Control Systems Magazine*, vol. 17, pp. 75-93, 1997.

[114] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural Networks for Control Systems-A Survey," *Automatica*, vol. 28, pp. 1083-1112, 1992.

[115] K. S. Narendra, "Neural Networks for Control: Theory and Practice," *Proceedings of the IEEE*, vol. 84, pp. 1385-1406, 1996.

[116] J. J. Hopfield, "Artificial Neural Networks," *Circuits and Devices Magazine*, vol. 4, pp. 3-10, 1988.

[117] G. A. Tagliarini, J. F. Christ, and E. W. Page, "Optimization Using Neural Networks," *IEEE Transactions on Computers*, vol. 40, pp. 1347-1358, 1991.

[118] F. Yuan, L. A. Feldkamp, and G. V. Puskorius, "Neural Networks in Automotive Control: Series Hybrid Electric Vehicle," *Benelux Quarterly Journal on Automatic Control*, vol. 37, pp. 11-16, 1996.

[119] D. H. Swan, M. Arikara, and A. D. Patton, "Battery Modeling for Electric Vehicle Applications Using Neural Networks," *SAE Paper 931009*, 1993.

[120] B. Baumann, G. Rizzoni, and G. Washington, "Intelligent Control of Hybrid Vehicles Using Neural Networks and Fuzzy Logic," *SAE Paper 981061*, 1998.

[121] S. R. Bhatikar, R. L. Mahajan, K. B. Wipke, and V. Johnson, "Artificial Neural Network Based Energy Storage System Modeling for Hybrid Electric Vehicles," *SAE Paper 2000-01-1564*, 2000.

[122] J. J. Hopfield and D. W. Tank, ""Neural" Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, pp. 141-152, 1985.

[123] R. Saeks and C. Cox, "Design of an Adaptive Control System for a Hybrid Electric Vehicle," presented at IEEE International Conference on Systems, Man, and Cybernetics, 1999.

[124] R. W. Schmitz, T. F. Wilton, and J. J. Anderson, "Method and Apparatus for Adaptive Energy Control of Hybrid Electric Vehicle Propulsion," in *U.S. Patent and Trademark Office Database*. United States: Transportation Techniques LLC., 2003.

[125] J. S. Albus, "Data Storage in the Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 63, pp. 228-233, 1975.

[126] J. S. Albus, "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement, and Control*, vol. 63, pp. 220-227, 1975.

[127] M. Brown and C. Harris, *Neurofuzzy Adaptive Modelling and Control*. New York: Prentice Hall, 1994.

[128] G. Burgin, "Using Cerebellar Arithmetic Computers," *AI Expert*, vol. 7, pp. 32-41, 1992.

[129] P. C. Parks and J. Militzer, "Improved Allocation of Weights for Associative Memory Storage in Learning Control Systems," *IFAC Design Methods of Control Systems*, pp. 507-512, 1991.

[130] T. W. Miller and F. H. Glanz, "Cerebellar Model Arithmetic Computer," in *Fuzzy Logic and Neural Network Handbook*, C. H. Chen, Ed. New York: McGraw-Hill, 1996.

[131] G. Strang, *Linear Algebra and Its Applications*, 3rd ed. Fort Worth, TX: Harcourt Brace Jovanovich College Publishers, 1988.

[132] P. C. Parks and J. Militzer, "Convergence Properties of Associative Memory Storage for Learning Control Systems," *Automation and Remote Control*, vol. 50, pp. 254-286, 1989.

[133] S. A. Lane, D. A. Handelman, and J. J. Gelfand, "Theory and Development of Higher-Order CMAC Neural Networks," in *Control Systems Magazine*, vol. 12, 1992, pp. 23-30.

[134] A. Brooker, K. Haraldsson, T. Hendricks, V. Johnson, K. Kelley, and B. Kramer, "ADVISOR Documentation," National Renewable Energy Laboratory, 2002.

[135] HEV-Center, "Visteon/UC-Davis HEV Simulation Documentation," University of California-Davis, Davis, CA 2000.

[136] J. D. Anderson, *Aircraft Performance and Design*. Boston, MA: McGraw-Hill, 1999.

[137] J. D. Anderson, *Introduction to Flight*, 2nd ed. New York: McGraw-Hill, 1985.

[138] R. Von Mises, *Theory of Flight*. New York: Dover Publications, 1959.

[139] D. P. Raymer, *Aircraft Design: A Conceptual Approach*. Reston, VA: AIAA, 1999.

[140] D. Stinton, *The Design of the Airplane*, 2nd ed. Reston, VA: AIAA, 2001.

[141] T. C. Corke, *Design of Aircraft*. Upper Saddle River, NJ: Prentice Hall, 2003.

[142] P. Gelhausen, "ACSYNT-A Standards-Based System for Parametric Computer Aided Conceptual Design of Aircraft," presented at 1992 Aerospace Design Conference, Irvine, CA, 1992.

[143] D. P. Raymer and W. A. Crossley, "Variations of Genetic Algorithm and Evolutionary Methods for Optimal Aircraft Sizing," presented at AIAA Aircraft Technology, Integration, & Operation Meeting, Los Angeles, CA, 2002.

[144] C. M. E. Holden, R. Davies, and A. J. Keane, "Optimization Methodologies in Conceptual Design," AIAA Paper 2002-5524, 2002.

[145] M. Ehsani, K. M. Rahman, and H. A. Toliyat, "Propulsion System Design of Electric and Hybrid Vehicles," *IEEE Transactions on Industrial Electronics*, vol. 44, pp. 19-27, 1997.

[146] J. W. Youngblood and T. A. Talay, "Solar-Powered Airplane Design for Long-Endurance, High-Altitude Flight," presented at AIAA 2nd International Very Large Vehicles Conference, Washington, DC, 1982.

[147] M. S. Selig, J. F. Donovan, and D. B. Fraser, *Airfoils at Low Speeds*. Virginia Beach, VA: H.A. Stokely, 1989.

[148] R. Eppler, *Airfoil Design and Data*. Berlin, Germany: Springer-Verlag, 1990.

[149] S. Siddiqi, R. Evangelista, and T. S. Kwa, "The Design of a Low Reynolds Number RPV," presented at Low Reynolds Number Aerodynamics, Notre Dame, IN, 1989.

[150] S. Weinzieri, R. Wildemann, and B. Hanula, "The Design and Development of a Light-Weight, High Speed, Diesel Engine for Unmanned Aerial Vehicles," *SAE Paper 2002-01-0160*, 2002.

[151] S. P. Dev, "JP-8/Battery Hybrid Propulsion and Power for Small UAVs and UGVs," presented at AUVSI's 30th Annual Unmanned Systems Symposium and Exhibition, Baltimore, MD, 2003.

[152] J. B. Heywood and E. Sher, *The Two-Stroke Cycle Engine: Its Development, Operation, and Design*. Warrendale, PA: SAE, 1999.

[153] J. B. Heywood, *Internal Combustion Engine Fundamentals*. New York: McGraw-Hill, 1988.

[154] "Mini 4-Stroke Engines," Honda, 2002.

[155] J. F. Manwell, J. G. McGowan, and A. L. Rogers, *Wind Energy Explained*. New York: John Wiley & Sons, 2002.

[156] J. S. Rohatgi and V. Nelson, *Wind Characteristics*. Canyon, TX: Alternative Energy Institute, 1994.

[157] "Wind Energy Engineering Tool Box of MiniCodes," Beta Test Version 1.01c ed. Amherst, MA: University of Massachusetts, 2000.

[158] J. P. Conner and A. S. Arena, "Advanced Dynamometer Designed to Fully Characterize the Propulsion System for a UAV," *SAE Paper 2002-01-2921*, 2002.

[159] J. K. Peterson, "The Cerebellar Model Articulated Controller (CMAC): Neural Architecture-Theory and Software Implementation." Clemson, SC: Department of Mathematical Sciences, 2002.

BLANK